



LIBRARY OF THE  
UNIVERSITY OF ILLINOIS  
AT URBANA-CHAMPAIGN

510.84

Il6r

no. 601-606

cop. 2



### CENTRAL CIRCULATION AND BOOKSTACKS

The person borrowing this material is responsible for its renewal or return before the **Latest Date** stamped below. **You may be charged a minimum fee of \$75.00 for each non-returned or lost item.**

Theft, mutilation, or defacement of library materials can be causes for student disciplinary action. All materials owned by the University of Illinois Library are the property of the State of Illinois and are protected by Article 16B of *Illinois Criminal Law and Procedure*.

**TO RENEW, CALL (217) 333-8400.**

**University of Illinois Library at Urbana-Champaign**

MAY 18 2000

When renewing by phone, write new due date  
below previous due date.

L162





Digitized by the Internet Archive  
in 2013

<http://archive.org/details/viptranaprogramm603weav>







Ill  
no. 603  
UIUCDCS-R-73-603

Math

VIPTRAN - A PROGRAMMING LANGUAGE AND ITS  
COMPILER FOR BOOLEAN SYSTEMS OR  
PROCESS CONTROL EQUATIONS

by

Alfred Charles Weaver

November 1973



DEPARTMENT OF COMPUTER SCIENCE  
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

THE LIBRARY OF THE

JAN 17 1974

UNIVERSITY OF ILLINOIS  
AT URBANA-CHAMPAIGN



UIUCDCS-R-73-603

VIPTRAN - A PROGRAMMING LANGUAGE AND ITS  
COMPILER FOR BOOLEAN SYSTEMS OR  
PROCESS CONTROL EQUATIONS

by

Alfred Charles Weaver

November, 1973

Department of Computer Science  
University of Illinois at Urbana-Champaign  
Urbana, Illinois

This work was supported in part by the Department of Computer Science and submitted in partial fulfillment for the Master of Science degree in Computer Science, 1973.



## ACKNOWLEDGMENT

The author wishes to express his gratitude to Professor T. A. Murrell, thesis advisor and hardware designer for the VIP, for his continuous consultation and support of the project, and for his contribution of the SORT and PLOTTER modules. A special thanks is also due Professor T. R. Wilcox, for his technical consultation on the art and science of compiler writing. Thanks are also given to Mrs. June Winger, whose patience and skill produced the typewritten manuscript.

Additional thanks go to the Department of Computer Science, University of Illinois at Urbana-Champaign, who financially supported this research and to Struthers-Dunn Company, Systems Control Division, Bettendorf, Iowa, whose consultation and use of the VIPTRAN compiler helped me formulate the VIPTRAN syntax and debug the compiler.



## TABLE OF CONTENTS

	Page
1. THE NEED FOR VIPTRAN PROGRAMMING.....	1
1.1 Introduction.....	1
1.2 The VIP Controller.....	1
1.3 VIP Programming Example 1.....	6
1.4 VIP Programming Example 2.....	9
1.5 VIP Programming Example 3.....	10
1.6 The VIPTRAN Compiler.....	14
2. VIPTRAN LANGUAGE DEFINITION.....	16
2.1 Definitions.....	16
2.2 VIPTRAN Compiler Options.....	20
2.3 VIPTRAN Syntax.....	22
2.3.1 Variable Type Declaration and Address Assignment....	22
2.3.1.1 Automatic Address Assignment Algorithm.....	24
2.3.1.2 Source Card Requirements.....	26
2.3.2 Control Program Segment.....	27
2.3.3 Advanced VIPTRAN - Control of X and Y registers.....	28
2.3.3.1 The X Register.....	29
2.3.3.2 The Y Register.....	31
2.3.3.3 X and Y Registers in Combination.....	32
2.3.3.4 The ON-OFF Variables.....	33



	Page
3. PROGRAM LOGIC MANUAL.....	34
3.1 Overview of the Compiler's Responsibilities.....	34
3.2 Module Technical Descriptions.....	36
3.2.1 BATCH.....	36
3.2.2 COMPILER.....	38
3.2.3 LEXI.....	40
3.2.4 LOOKUP.....	42
3.2.5 MAPPING.....	42
3.2.6 ERROR.....	43
3.2.7 SYNA.....	43
3.2.8 ADCHECK.....	43
3.2.9 EXPRESSION.....	44
3.2.10 OUT.....	44
3.2.11 DUMP.....	44
3.2.12 SEMA .....	46
3.2.13 SORT.....	47
3.2.14 CODEGEN.....	48
3.2.15 TEMPGEN.....	51
3.2.16 GEN.....	52
3.2.17 OBJECT.....	53
3.2.18 PRINT .....	53
4. SUMMARY.....	54
LIST OF REFERENCES.....	55
APPENDIX	
A. VIPTRAN Programming Examples.....	56
B. VIPTRAN Language Syntax, Modified Backus-Naur Form.....	117
C. VIPTRAN Error Messages.....	120



## LIST OF FIGURES

Figure	Page
1. Relay Diagram for Burglar Alarm Example.....	8
2. VIPTRAN Compiler Module Containment Map.....	37



## 1. THE NEED FOR VIPTRAN PROGRAMMING

### 1.1 Introduction

VIPTRAN is a high-level, FORTRAN-like programming language specifically designed for use with the Struthers-Dunn VIP (Variable Industrial Programmer), a multi-purpose industrial process control minicomputer. The language allows the programmer to express a Boolean system of control parameters as a sequence of semi-Boolean equations whose cyclic evaluation in real time determines the on-off state of real world devices.

Previous attempts to solve the process control problem have followed the general scheme of expressing process control logic by some series of pseudo-assembler language commands which are then translated into a system simulation language, analyzed and processed at this higher level, then retranslated into machine understandable commands. Attempts to perform this operation in real time generally result in either slow response times or extra expense for faster-than-necessary computing circuitry. VIPTRAN and the VIP in combination allow the programmer the ease and flexibility of expressing algorithms in a higher-level language while dispensing with all the system simulation by merely solving the Boolean equations themselves.

### 1.2 The VIP Controller

To understand adequately the features and capabilities of VIPTRAN, it is necessary first to investigate the nature of the VIP itself. The

VIP is actually an electronically programmable logic system, designed to perform the same control functions as a conventional relay or solid state logic system. Information is accepted at the inputs of the VIP, analyzed, and the appropriate output commands provided. The input information may be in the form of voltage or current levels from such devices as limit switches, pushbuttons, photo-electric controls, etc. Output commands are provided from the controller in the form of voltage or current signals capable of driving conventional solenoids, relays, lights, motor starters, and similar devices. Inputs are generally classified as 6, 12, 24, 48, or 120 volt AC or DC inputs. The existence of a zero voltage is considered a logic "0" (Boolean "false") while the non-zero voltage is considered a logic "1" (Boolean "true").

External inputs are connected to the VIP through input register circuits. Information fed to the VIP in the form of voltage levels represents the logical condition of the external switches. If a given switch is closed, the input register stores a logic "1"; if open, the input register stores a logic "0".

The output register is the means by which all command signals are fed from the VIP to the real world. The output registers actually serve two purposes:

- (1) they provide a contact closure for each load (motor, solenoid, light, etc.) in the form of a solid state switch;
- (2) information stored within the output register corresponding to a particular device is representative of that device's physical condition, i.e., energized or de-energized.

Output registers may be either AC or DC.

Timer registers may be used in place of an output register to introduce a delay between the storing of a logic "1" and the energizing of the external load. This delay is physically adjustable and covers a range of approximately 100 milliseconds to 30 seconds.

Magnetic latch memories are also interchangeable with output registers. In the event of a power failure to the VIP these retentive memories retain their most recent value, thus duplicating the memory inherent in the mechanical or magnetic two-coil latching relay.

A scratchpad memory is also available which provides a storage area for temporary or virtual results. One scratchpad contains 512 addressable locations.

The input, output, timer, and latch registers are available in groups of sixteen like devices on one plug-in "card". The physical location of these plug-in cards thus forms the address space used by the machine. If, for instance, one were to place one each of input, output, timer, and latch cards in the successive low-order card positions, the address space would be:

<u>address space</u>			
<u>card position</u>	<u>card type</u>	<u>decimal</u>	<u>octal</u>
1	24-volt DC input	0 through 15	0 through 17
2	DC output	16 through 31	20 through 37
3	timer	32 through 47	40 through 57
4	latch	48 through 63	60 through 77

Each plug-in card thus occupies  $16_{10}$  contiguous addresses beginning at an address which is a multiple of  $16_{10}$ . Each card is positionally interchangeable with all others, except for the low-order card (addresses 0

through  $17_8$ ) which must be of type input. Location 0 is reserved; line power to the VIP is connected here, thus forming the basis for a guaranteed logic "1" at address 0.

Scratchpad memory is slightly different in that it is one card which may be inserted or deleted as a matter of convenience. If included, it provides  $512_{10}$  additional "scratch" memory locations (or "virtual control relays") at addresses  $1000_8$  through  $1777_8$ .

The VIP is equipped with a 4096-word by 8-bit read-only memory in which the control program is stored. Depending upon the total number of inputs, outputs, timers, and latches used by the control program, the VIP is available in three models:

mini-VIP	$128_{10}$	addressable locations
midi-VIP	$256_{10}$	addressable locations
maxi-VIP	$512_{10}$	addressable locations

Each model will accommodate a single optional scratchpad.

When the control program is executed, the following sequence of events is performed cyclically:

- (1) At the beginning of each scan of the read-only memory, and synchronized with the peak voltage of the power line, all of the inputs to the machine are simultaneously examined and their logic state stored in the input holding registers. Thus all inputs which change state do so simultaneously at the beginning of each memory scan.
- (2) The entire logical sequence of instructions ( $\leq 4096$  instructions) is executed sequentially at a rate of one instruction per ten microseconds, or 40.96 milliseconds to scan all of memory once.

- (3) At the end of the memory scan, and synchronized with a zero voltage on the power line, all of the outputs in the output holding registers are simultaneously gated to the output terminals. Thus all outputs which change state do so simultaneously at the end of each memory scan.

The above sequence of operations is repeated indefinitely.

The instruction set of the VIP consists of the following mnemonic operation codes and operands:

LDA <address>	load accumulator with data at <address>
LDAC <address>	load accumulator with the complement of the data at <address>
AND <address>	Boolean AND the accumulator with the data at <address>, leaving the result in the accumulator
ANDC <address>	Boolean AND the accumulator with the complement of the data at <address>, leaving the result in the accumulator
OR <address>	Boolean OR the accumulator with the data at <address>, leaving the result in the accumulator
ORC <address>	Boolean OR the accumulator with the complement of the data at <address>, leaving the result in the accumulator
STO <address>	store the content of the accumulator into <address> (accumulator is unchanged)
AUX 777	no operation (erasure)
AUX 776	load the X register from the accumulator
AUX 775	load the Y register from the accumulator
AUX 774	the 8-bit address of the next sequential instruction refers to scratchpad

AUX 773                   all forthcoming 8-bit addresses refer to  
                           scratchpad until an AUX 772 instruction is  
                           processed

AUX 772                   terminate scratchpad addressing mode

AUX 771                   end of program

The CPU of the VIP contains one 1-bit accumulator for evaluating logical expressions, one 1-bit X register whose content is always ANDed with the accumulator and the resulting quantity stored for each STO instruction (the accumulator itself remains unchanged), and one 1-bit Y register which, if zero, inhibits the action of STOR instructions, thus simulating a logical "jump" around a block of code.

At this point three examples will help to clarify the coding procedure.

### 1.3 VIP Programming Example 1

A bank has installed a VIP to sound a burglar alarm and turn off electrical power to the vault door timer in any of the following events:

- (1) The bank office door is opened while the burglar alarm power switch is activated;
- (2) The vault door is opened while the burglar alarm power switch is activated;
- (3) A manual pushbutton is depressed.

It is natural to invent Boolean variables to represent the logical states as described above. Let us use the following:

<u>variable name</u>	<u>type</u>	<u>description</u>
DOOR	input	true when bank door is open, false when bank door is closed
SWITCH	input	true when power is applied to burglar alarm, false when power is not applied

<u>variable name</u>	<u>type</u>	<u>description</u>
VAULT	input	true when vault door is open, false when vault door is closed
PUSHBUTTON	input	true when pushbutton is depressed (closed), false when pushbutton is released (open)
ALARM	output	true when alarm is to sound, false when alarm is to be silent
TIMER	output	true when power is applied to vault door timer, false when power is interrupted

The two Boolean control equations, without simplification, are:

$$\text{ALARM} = (\text{DOOR} \wedge \text{SWITCH}) \vee (\text{VAULT} \wedge \text{SWITCH}) \vee \text{PUSHBUTTON}$$

$$\text{TIMER} = (\text{DOOR} \wedge \text{SWITCH}) \vee (\text{VAULT} \wedge \text{SWITCH}) \vee \text{PUSHBUTTON}$$

where  $\wedge$  represents a Boolean AND,

$\vee$  represents a Boolean OR,

and  $\text{---}$  represents a Boolean complement (NOT).

The system, expressed as a relay tree, might be graphically depicted as in Figure 1.

Let DOOR, SWITCH, VAULT, and PUSHBUTTON be inputs to a VIP, connected to input register locations  $1_8$ ,  $2_8$ ,  $3_8$ , and  $4_8$  respectively. Let ALARM and TIMER be outputs from the VIP, connected to an AC output register at locations  $20_8$  and  $21_8$  respectively. The machine code for the VIP would be:

<u>instruction</u>	<u>address</u>	<u>variable name</u>	<u>comments</u>
LDA	1	DOOR	load status of door
AND	2	SWITCH	AND with alarm switch status

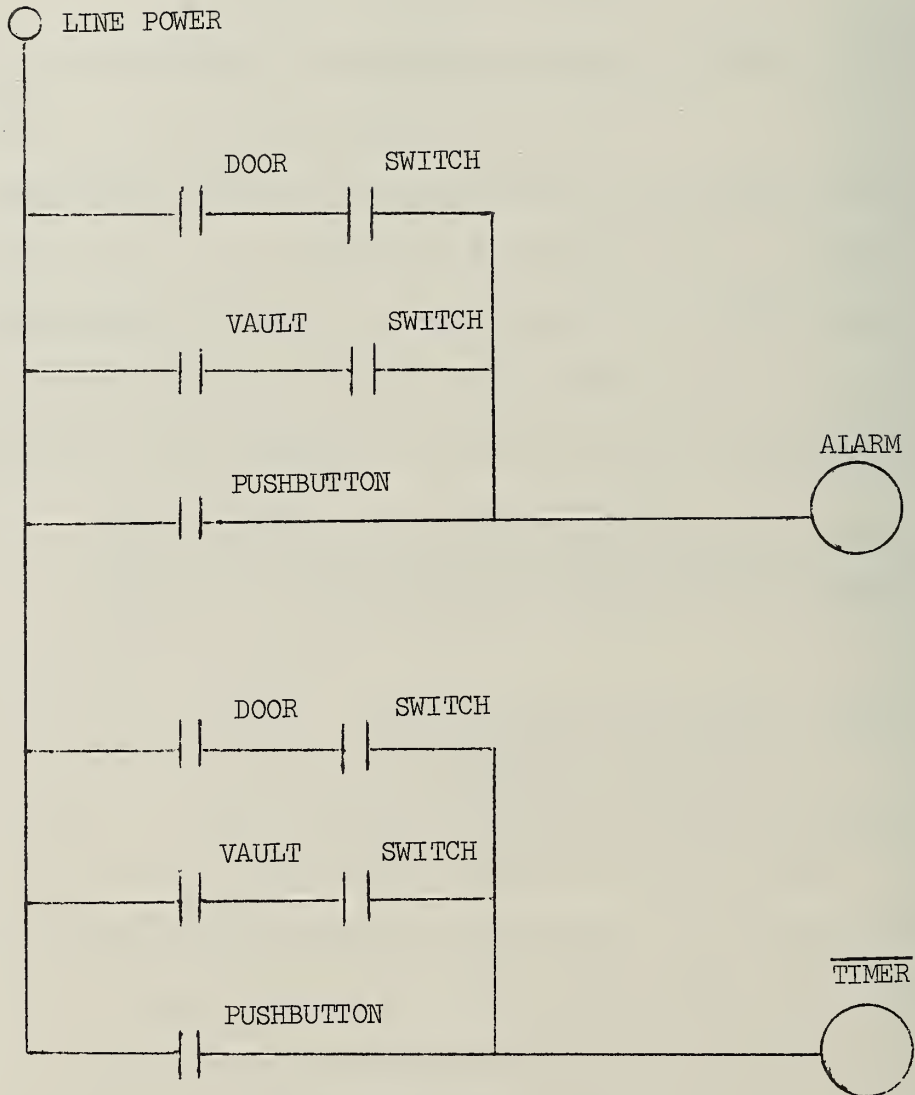


Figure 1. Relay Diagram for Burglar Alarm Example

<u>instruction</u>	<u>address</u>	<u>variable name</u>	<u>comments</u>
STO	22	TEMP1	save result in temporary location
LDA	3	VAULT	load vault status
AND	2	SWITCH	AND with alarm switch status
OR	22	TEMP1	OR with temporary result
OR	4	PUSHBUTTON	OR with button status
STO	20	ALARM	output status of alarm
LDA	1	DOOR	load status of door
AND	2	SWITCH	AND with alarm switch status
STO	22	TEMP1	save temporary result
LDA	3	VAULT	load vault status
AND	2	SWITCH	AND with alarm switch status
OR	22	TEMP1	OR with temporary result
OR	4	PUSHBUTTON	OR with button status
STO	23	TEMP2	save temporary result
LDAC	23	TEMP2	load complement of TEMP
STO	21	TIMER	output status of timer

#### 1.4 VIP Programming Example 2

Of course the preceding example made no attempt to optimize either the Boolean equations or the VIP code for the system. The application of simple Boolean identities reduces the control equations to

$$\text{ALARM} = ( (\text{DOOR} \vee \text{VAULT}) \wedge \text{SWITCH} ) \vee \text{PUSHBUTTON}$$

$$\text{TIMER} = \overline{\text{ALARM}}$$

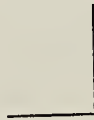
which codes as follows:

<u>instruction</u>	<u>address</u>	<u>variable name</u>	<u>comments</u>
LDA	1	DOOR	load status of door
OR	3	VAULT	OR with vault status
AND	2	SWITCH	AND with alarm status
OR	4	PUSHBUTTON	OR with button status
STO	20	ALARM	output status of alarm
LDAC	20	ALARM	load complement of alarm
STO	21	TIMER	output timer status

Although both sets of equations are logically correct, their VIP implementation is much more efficient in the latter case.

### 1.5 VIP Programming Example 3

An automatic welder is to produce an L-shaped weld such as:



under the control of a VIP. One welding cycle, which is signaled by a momentary START switch, should:

- (1) start a motor for horizontal movement to the right and turn on power to the welder for 10 seconds;
- (2) then stop the horizontal motion and begin upward motion for 15 seconds;
- (3) then turn off power to the welder and return the welder to its physical origin. The origin is identified by two limit switches in the lower left corner being ON simultaneously;
- (4) the welder should wait at the origin until the start button is again depressed.

This problem introduces the concept of a timer and a timer function.

Any timer has two "sides" - input and output. A "1" stored into a timer address starts the timing process; storing a "0" has no effect. A load (LDA) from a timer address fetches the current output of the timer. Thus it is impossible to tell by only fetching whether a timer is "timed out" or whether it is off because it was never set. A second (scratch) variable is often used with timers to indicate the current input to the timer. The six available timer functions, using the notation of relay circuits, are:

<u>timer function</u>	<u>explanation</u>
T-OOX	output of timer T
T-OXO	input to timer T ANDed with complement of timer output
T-OXX	input to timer T
T-XOO	complement of input to timer T
T-XOX	complement of input to timer T ORed with output of timer T
T-XXO	complement of output of timer T

The "code" for the timer function is:

-abc

where a is the condition of the contact before timing,

b is the condition of the contact during timing,

c is the condition of the contact after the timer has "timed out,"

and a, b, and c represent the contact status:

X = closed,

O = open.

Using the following variable names and address assignments:

<u>variable name</u>	<u>address</u>	<u>type</u>	<u>identification</u>
LEFT	20	output	motor control, horizontal movement to the left
RIGHT	21	output	motor control, horizontal to the right
UP	22	output	motor control, upward movement
DOWN	23	output	motor control, downward movement
WELD	24	output	power to welder
LS1	1	input	limit switch 1, horizontal
LS2	2	input	limit switch 2, vertical
START	3	input	momentary pushbutton switch
T10	40	timer	externally set for 10 seconds
T15	41	timer	externally set for 15 seconds

The strategy should be:

- (1) activate LEFT when both T10 and T15 have timed out and LS1 is off;
- (2) activate DOWN when both T10 and T15 have timed out and LS2 is off;
- (3) activate T10 (set timer) when START is on;
- (4) activate RIGHT while T10 is timing;
- (5) activate T15 (set timer) when T10 has has timed out;
- (6) activate UP while T15 is timing and RIGHT is off;
- (7) activate WELD when either RIGHT or UP is activated.

One (non-unique) set of Boolean control equations for this system

is:

$$\text{LEFT} = \text{T10-00X} \wedge \text{T15-00X} \wedge \overline{\text{LS1}}$$

$$\text{DOWN} = \text{T10-00X} \wedge \text{T15-00X} \wedge \overline{\text{LS2}}$$

$$\text{T10} = \text{START}$$

$$\text{RIGHT} = \text{T10-XX0}$$

$$\text{T15} = \text{T10-00X}$$

$$\text{UP} = \text{T15-XX0} \wedge \overline{\text{RIGHT}}$$

$$\text{WELD} = \text{LEFT} \vee \text{RIGHT}$$

The VIP machine code for this set of equations is:

<u>instruction</u>	<u>address</u>	<u>variable name</u>	<u>comments</u>
LDA	40	T10	load T10 output
AND	41	T15	AND with T15 output
ANDC	1	LS1	AND with complement of LS1
STO	20	LEFT	controls left movement
LDA	40	T10	load T10 output
AND	41	T15	AND with T15 output
ANDC	2	LS2	AND with complement of LS2
STO	23	DOWN	control down movement
LDA	3	START	load status of start button
STO	4	T10	possibly set T10
LDAC	40	T10	load T10 output complemented
STO	21	RIGHT	control right movement
LDA	40	T10	load T10 output
STO	41	T15	possibly set T15
LDAC	41	T15	load complement of T15 output
ANDC	21	RIGHT	AND with complement of RIGHT

<u>instruction</u>	<u>address</u>	<u>variable name</u>	<u>comments</u>
STO	22	UP	control up movement
LDA	20	LEFT	load left movement status
OR	21	RIGHT	OR with right movement status
STO	24	WELD	WELD is on if LEFT or RIGHT is on

## 1.6 The VIPTRAN Compiler

The foregoing examples begin to illustrate the desirability of a higher-level language rather than the currently used sequences of operation codes and addresses (this is equivalent to an elementary assembler language). VIPTRAN provides this flexibility by allowing the user to express his system of control equations in a high-level FORTRAN-like language designed especially for Boolean systems and the VIP. VIPTRAN allows the programmer to write arbitrarily complex assignment statements using the Boolean operators AND, OR, and NOT, the six timer functions T-OOX, T-OXO, T-OXX, T-XOO, T-XOX, and T-XXO, and two special operators which allow the programmer to optimize his own code by efficient use of the X and Y registers.

In addition, the VIPTRAN compiler will optionally perform all of the bookkeeping operations such as variable address assignment, VIP plug-in card assignment, and automatic addressing in scratchpad (when a scratchpad is included) or in unused outputs (when a scratchpad is not included) for compiler generated temporary variables (such as TEMP1 and TEMP2 in example 1). The compiler will also produce a memory map of the plug-in cards, showing their type and octal address, as well as a memory map of all individual variable

names, showing their type and address, sorted into ascending order by octal address.

Another option causes the compiler to sort all of the equations and thus find dependencies which are sensitive to physical location. If A, B, and C are variables used in a program, then the (trivial) sequence of assignment statements

$$A = B$$

$$B = C$$

should be reordered as

$$B = C$$

$$A = B$$

so that A will have a correct value on the first scan of memory. The sorting algorithm identifies all such interdependencies and reorders the equations to eliminate as many as possible. In the event that two or more equations are unresolvably interlocked, as in

$$A = B \wedge C$$

$$B = \overline{A} \vee C$$

the compiler identifies all such equations for further study by the programmer.

All "normal" program errors are trapped by the compiler and an explicit, informative error message is produced, physically located near the site of the error itself on the source listing.

The object code produced by the compiler is printed showing the read-only memory word number, instruction mnemonic, octal address, and variable name. If desired, a companion program (PLOTTER) can use a disk copy of the compiler's output to produce a relay tree depicting the controlled system in conventional relay logic graphics. Other features of the compiler allow the programmer maximum flexibility in his effort to turn Boolean equations into VIP machine code.

## 2. VIPTRAN LANGUAGE DEFINITION

### 2.1 Definitions

The VIPTRAN language syntax, technically described in modified Backus-Naur Form in Appendix B, is best understood after some basic definitions of commonly used terms.

**VARIABLE** - a logical quantity which can assume the Boolean values of "1" and "0" (or, equivalently, "true" and "false").

A variable always assumes one of its two possible values.

**VARIABLE NAME** - a character string which is the symbolic reference to a variable. In a previous example, VAULT was the variable name which referred to the logical condition of a bank vault door being either open or closed. Every variable name has an associated octal address corresponding to the physical layout of the VIP. Again using the bank vault example, VAULT was the variable name which was used to denote the logic variable stored at input address 3<sub>8</sub>. All variable names contain 1 to 12 contiguous characters. (Note: Only the first 8 characters of the name will be reproduced by PLOTTER.) The legal characters for a variable name are:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z " . - 0 1 2 3 4 5 6 7 8 9

The first character must not be a digit (0 through 9). Legal variable names include:

A

ABC

VAULT

"VAULT"

ZZ1234567890

T-1-2

The following are illegal variable names:

6	first character is a digit
2B	first character is a digit
A B	characters are not contiguous
NAMEISTOOLONG	contains more than 12 characters
OFF?	illegal character ?

ADDRESS - the physical location of a variable. If the associated variable is an input or output variable, the address also specifies the physical location of the external device connection. All addresses are specified in octal. All addresses are in the range  $0 \leq \text{address} \leq 1777_8$ . Addresses 0 and  $1000_8$  are reserved and cannot be used by the programmer.

FUNCTION - a specific combination of a timer's input and output.

The six timer functions may be applied to any legal variable name which has been declared to be of type timer by appending one of the following functional suffixes to the timer name:

<u>function suffix</u>	<u>function value</u>
-OOX	output of timer
-OXO	input to timer ANDed with complement of timer output
-OXX	input to timer
-XOO	complement of input to timer

<u>function suffix</u>	<u>function value</u>
-XOX	complement of input to timer ORed with output of timer
-XXO	complement of output of timer

Examples:

T6-XOX	legal if T6 is a timer
TIMER-6-OOX	legal if TIMER-6 is a timer
T6-OYX	illegal if T6 is a timer
INTERVALTIM-XOO	illegal; more than 12 characters
-OOX	legal variable name (not a timer function)
-XXX	legal variable name (not a timer function)

The first three examples could be legal variable names themselves (not functions) if T6 and TIMER-6 are not variable names of type timer.

OPERATOR - one of three Boolean operations.

<u>operator</u>	<u>symbol</u>	<u>operator type</u>
AND	*	binary
OR	+	binary
NOT	/	unary

As in algebra, the operators have an inherent priority so that any string of variables and operators will have one and only one logical interpretation (i.e., the meaning is unambiguous). The order of evaluation of terms in a VIPTRAN expression is

1. terms inside the innermost parentheses
2. unary NOT
3. binary AND
4. binary OR
5. left to right

TERM - a complemented or uncomplemented variable name or timer function.

EXPRESSION - the result of ANDing and ORing Boolean terms, using parentheses to control grouping and to distribute the unary NOT over several terms.

<u>Boolean expressions</u>	<u>equivalent VIPTRAN expression</u>	
$A \vee B$	$A + B$	
$A \wedge B$	$A * B$	
$A \wedge (B \vee C)$	$A * (B + C)$	parentheses necessary
$A \vee (B \wedge C)$	$A + B * C$	parentheses unnecessary because AND has priority over OR
$\bar{A} \vee B$	$/A + B$	NOT has priority over OR
$\overline{A \vee B}$	$/(A + B)$	parentheses necessary to distribute NOT
$A \vee ((\overline{B \wedge C}) \wedge D)$	$A + /(B * C) * D$	parentheses necessary to distribute NOT
$(A \wedge B) \vee (\bar{A} \wedge \bar{B})$	$A * B + /A * /B$	parentheses unnecessary
$\overline{A \vee B \vee C \vee D}$	$/(A + /(B + /(C + D)))$	parentheses necessary

Extra parentheses in the VIPTRAN expression cause no logic problem; thus  $(A)*(B*(C*D)) \equiv A*B*C*D$ . Furthermore, when the nature of the expression requires a temporary result to be generated, as in  $(A+B) * (C+D)$ , the compiler automatically allocates an unused variable to store the result of  $A+B$  while  $C+D$  is being evaluated.

ASSIGNMENT STATEMENT - the specification of where the result of an evaluated expression should be stored. It is of the form

$\langle \text{variable} \rangle = \langle \text{expression} \rangle$

No blanks are required around the equal sign or throughout the expression. Note that only variable names and timer names (not timer functions) may appear to the left of the equal sign.

DECLARATION - the association of a specific variable type (input, output, timer, latch, scratch) with a specific variable name.

## 2.2 VIPTRAN Compiler Options

The first card of a VIPTRAN program is the \$VIP card, so called because it contains the characters \$VIP in card columns 1 through 4, inclusive. Any cards preceding the \$VIP card will be completely ignored, and thus may be freely used for comments or any other purpose whatsoever. The \$VIP card is used to specify which compiler options the user wishes to invoke. For each of 9 groups of options, one of the choices is already the default option. Explicitly stating the default option has no effect, while stating another option within the group resets the compiler's logic to perform (or not perform) the specified task. The 9 options, with their default values underlined, are:

SOURCE or NOSOURCE

The SOURCE option causes the input deck to be listed in its entirety, with card numbers and program statement numbers added for easy reference. Most error messages refer to a particular card number or program statement number, so un-debugged programs should use the SOURCE option until they are error-free. After this point, and particularly if the source deck is long, NOSOURCE prevents needless repetition and saves some execution time.

SORT or NOSORT

The SORT option causes the compiler to examine and possibly reorder the given sequence of source equations. The sorting algorithm will determine whether the equations are interlocked, and if so will attempt to move one or more equations to new positions which will make them independent. If one or more groups of equations are interlocked and cannot be satisfactorily rearranged, the sort fails and a message describing the situation is printed

PRINTSORT or NOPRINTSORT

The PRINTSORT option will cause the source equations to be listed in their (new) sorted order - the order from which the compiler will generate code; NOPRINTSORT suppresses this listing. NOSORT implies NOPRINTSORT.

CODE or NOCODE

The CODE option causes the compiler to generate object code for the VIP if there are no serious errors in the program. NOCODE inhibits the code generation phase and allows fast syntax checking with a significant decrease in execution time.

SORTFAIL or NOSORTFAIL

If the SORT option has been specified, the SORTFAIL option will cause the compiler to stop before code generation if the sorter has found inseparable dependencies among the equations; NOSORTFAIL allows the code generator to proceed without regard to any error messages produced by the sorter.

MAP or NOMAP

The MAP option lists all the variable names, their types, and their octal addresses in order of ascending octal address; NOMAP inhibits the listing.

TTY or NOTTY

If the compiler's output is to be printed by a teletype, option TTY should be in use. If the output is to be printed by a line printer, NOTTY should be specified.

SCRATCHPAD or NOSCRATCHPAD

SCRATCHPAD specifies that a VIP scratchpad card will be inserted in the machine and is available to the compiler for compiler-generated temporary

variables. NOSCRATCHPAD indicates that no scratchpad will be included in this machine.

SKIPd - d represents 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. Default: d = 0.

After every STO instruction, d VIP read-only memory words are skipped (left blank), thus leaving small "holes" in the memory for subsequent error correction or code modification.

## 2.3 VIPTRAN Syntax

The VIPTRAN language syntax can be divided into two distinct segments: variable type declaration and optional address assignment, and the Boolean equations forming the control program itself. Let us discuss the two segments separately.

### 2.3.1 Variable Type Declaration and Address Assignment

Any VIPTRAN variable is one of ten possible types. The types and their corresponding VIPTRAN keyword abbreviations are:

<u>physical type</u>	<u>VIPTRAN declaration keyword</u>
6 volt input	INPUT6.
12 volt input	INPUT12.
24 volt input	INPUT24.
48 volt input	INPUT48.
120 volt input	INPUT120.
AC output	OUTPUTAC.
DC output	OUTPUTDC.
magnetic latch retentive memory	LATCH.
timer	TIMER.
scratchpad	SCRATCH.

To declare any list of variables to be one of the above types, the declaration keyword is listed, followed by the list of variables to be so typed, with each variable and declaration keyword separated from its neighbor by one or more blanks. When all of the variables of a given type have been listed, a new declaration keyword is listed followed by its own list of variables names. The process is repeated, using as many declarations and as many lines as necessary, until all variables which need to be typed have been declared.

Example:

```
INPUT6. A1 B2 C3
```

```
OUTPUTAC. OUT1 OUT2
```

```
LATCH. CR1 SCRATCH. S1 S2 S3
```

The above example declares variables A1, B2, and C3 to be 6-volt inputs, OUT1 and OUT2 to be AC outputs, CR1 to be a magnetic latch, and S1, S2, and S3 to be in scratchpad. There is no required ordering of the declaration keywords. Each keyword may appear any number of times and may be followed by a null list. The following declaration segment is also correct.

Example:

```
INPUT120. OUTPUTDC. A B
```

```
INPUT120. IN6 INPUT120. IN7
```

```
INPUT6. D E F SCRATCH. S1
```

An exception to the above syntax is the "TIMER." declaration. Because four of the six timer functions require knowledge of the name of the variable in which the programmer is storing the timer's latest input, the syntax for timer declarations is: TIMER. keyword, followed by a left parenthesis, followed by the timer name, followed by a right parenthesis. Any number of timers may be declared in the list for a single declaration keyword. Only a single blank after the keyword is required by the syntax.

Example:

```
TIMER.  T1 (T1INPUT)  T2 (RELAY6)
          T3 ( CR46 )  T4 ( T4INPUT )
```

For each of the above examples, the octal address of the variable will be determined and mapped by the compiler. This option may be overridden if the user desires to specify the address of one or more variables by applying the following rule:

Wherever a variable name appears in a declaration, it may be optionally followed by an octal address, separated from the variable name by one or more blanks. Those variable names followed by legal octal addresses will be assigned to those physical locations; all other variables will be optimally assigned by the compiler to fit "around" those (if any) specified by the user.

#### 2.3.1.1 Automatic Address Assignment Algorithm

The following strategy is used by the compiler to allocate the octal address of unspecified variables:

- (1) If the programmer supplies an address,
  - (a) the address is error-checked for being an octal number, for being in the proper range, and for not being multiply defined;
  - (b) if the variable is type "scratch" its address must be in the range  $1000_8 < \text{address} \leq 1777_8$ ;
  - (c) if the variable is not of type "scratch" its address must be in the range  $0 < \text{address} \leq 777_8$ ;
  - (d) the addresses 0 and  $1000_8$  are reserved and may be assigned by the compiler but not by the programmer.

- (2) If the programmer does not specify an address
  - (a) and if the variable is declared to be a specific type, the next available location on a plug-in card of that type will be assigned.
  - (b) and if the variable is not declared,
    - (i) and if the SCRATCHPAD option on the \$VIP card is in use, the next available scratchpad location is assigned.
    - (ii) and if the SCRATCHPAD option is not in use, the next unused position on any OUTPUTAC or OUTPUTDC card is assigned.
- (3) If the compiler must assign temporary variables for use during the code generation phase of compilation, as in the case of generating code to evaluate  $(A+B)*(C+D)$ , then
  - (a) if the SCRATCHPAD option is in use, the compiler assigns the next available scratchpad location.
  - (b) if the SCRATCHPAD option is not in use, the compiler assigns the next unused location on any OUTPUTAC or OUTPUTDC card. If all OUTPUTAC and OUTPUTDC cards are already full, the compiler generates an error message for lack of sufficient space and suggests re-running the program with the SCRATCHPAD option enabled.

When the compiler assigns a temporary variable, that variable (and hence its location) are reserved for the duration of the equation and will not be reused, even if it is "safe" to do so. This feature aids debugging of machines running in the field and allows examination of temporary results. Compiler generated temporaries are given the mnemonic name `TEMPddd` where `ddd` is the new variable's octal address.

After all of the code for an equation has been generated, all temporaries used by that equation are "released" and will be reused in the next equation requiring temporary storage.

In all of the cases listed above except 3(b), the allocation of a variable of some type will use a position on an unfilled card of that type before allocating an additional card ( $16_{10}$  variables) of that same type. Situation 3(b) is handled optimally since the allocation of an additional OUTPUTAC or OUTPUTDC card, with its  $16_{10}$  variable capacity, is just as expensive as adding a scratchpad with a capacity of  $512_{10}$  variables. Re-running the program with the SCRATCHPAD option enabled might well eliminate one or more (expensive) OUTPUTAC or OUTPUTDC cards.

The user may specify some addresses and allow the compiler to assign all of the others. A variable may be declared more than once as long as no type or address conflicts occur.

Example:

```
INPUT6.  A 2   B   C   D 3   E
OUTPUTDC.  X   Y 100   Z 110
TIMER.  T1 (A)   T2 40 ( B )
        T3 ( C 4 )   T4 41(D 3)
```

### 2.3.1.2 Source Card Requirements

Finally, each line of coding in the declaration segment represents one IBM card in terms of input to the compiler. Only columns 1 through 72 may be used, and column 1 is reserved for special purposes. Thus the declaration segment text is contained within columns 2 through 72, inclusive and may occupy any number of sequential cards. The source program listing will show each of the source cards sequentially numbered for ease of reference and fast identification of errors.

### 2.3.2 Control Program Segment

The control program segment is identified by the declaration keyword "PROGRAM.". All of the assignment statements which make up the program logic follow the PROGRAM. keyword. All of the assignment statements must be contained within columns 2-72 of any one card. If an expression is too long to fit on one card, it may be continued to another card by placing the continuation character ">" in column 1 of the card which is the continuation. This makes columns 2 - 72 of the continuation card a logical extension of the previous card, subject to the constraint that variable names may not be "broken" across a card boundary. Continuation cards may themselves be continued by placing the continuation character in column 1 of the next continuation card. There is no limit to the number of continuation cards used to express a single assignment statement. There is no limit on the number of cards used to express the control program segment. There is a compiler-defined limit of  $512_{10}$  statements in any one program. The physical end of the control program segment is marked by a \$TRANSLATE card, containing those 10 characters in columns 1 to 10 of the card, and followed by a \$END card, containing those 4 characters in columns 1 to 4 of that card.

Documentation is an important part of any program. Placing the character "C" in column 1 of any card makes the entire card a comment. Any text on the comment card is printed on the source listing, but is in no way examined by the compiler. Comment cards may be used anywhere within a program except between continuation cards.

VIPTRAN is a batch compiler and can process any number of separate programs sequentially. Each program, from \$VIP card through \$END card inclusive, is stacked in the input stream and the compiler runs each program

independently and sequentially. Errors in any one program have no effect on the performance of any other program.

This degree of explanation of the VIPTRAN language is sufficient to allow the programmer to write a simple VIPTRAN program. There are some advanced features yet to be discussed which allow the user to optimize his programming, but first consider the programming examples in Appendix A.

2.3.3 Advanced VIPTRAN - Control of X and Y Registers

A common reality in process control is the recurrence of a specific variable or expression which expresses some "master" condition. A simple example is that a group of control equations might contain the terms

$$" * \text{LINEPOWER} "$$

where LINEPOWER is an input variable name describing the master on-off state of the controlled machine. In this instance, the control equations might be of the form

$$\begin{aligned} \text{LIGHT} &= \text{RUN} * / \text{START} * \text{LINEPOWER} \\ \text{GATE} &= (\text{RUN} + \text{MANUAL}) * \text{LINEPOWER} \\ \text{BELL} &= (/ \text{SAFETY} + / \text{GATE}) * \text{LINEPOWER} \end{aligned}$$

This example produces the following code:

<u>instruction</u>	<u>variable name</u>
LDA	RUN
ANDC	START
AND	LINEPOWER
STO	LIGHT
LDA	RUN
OR	MANUAL
AND	LINEPOWER
STO	GATE

<u>instruction</u>	<u>variable name</u>
LDAC	SAFETY
ORC	GATE
AND	LINEPOWER
STO	BELL

### 2.3.3.1 The X Register

While the program and code given are both exactly correct, an obvious optimization, considering the hardware capabilities of the VIP, would be to use the X register to hold the repeated variable LINEPOWER. Since the X register is automatically ANDed with the accumulator before a STORE instruction is executed, loading the X register with LINEPOWER would have the same logical effect as repeating "AND LINEPOWER" in the instruction stream for each assignment statement. There would be a net saving of one instruction per equation less the two instructions necessary to load LINEPOWER into the accumulator and then load the X register from the accumulator.

For this purpose, VIPTRAN uses a fourth binary operator, the ampersand (&). The definition of the "&" operator is that it loads the X register with the contents of the variable name immediately to its left, if and only if that variable is not already stored in the X register. By replacing "\* LINEPOWER" with "LINEPOWER &" in the previous example, the following equations are generated:

LIGHT = LINEPOWER & RUN \* /START

GATE = LINEPOWER & (RUN + MANUAL)

BELL = LINEPOWER & (/SAFETY + /GATE)

The priority of "&" is greater than the other operands, so the loading of the X register is the first code generated for an assignment statement using "&".

Naturally, the "master control" of a set of equations may depend upon the logical combination of many variables, so an expression is generated to express that combination of conditions. The simplest way to utilize the X register optimization technique in this event is to assign the controlling expression to a "scratch" variable (for which the compiler will automatically assign an address if not specified by the programmer), then use the new variable as the object of the & operator.

Example:

```

MASTER = LINEPOWER * (LIMIT.SW1 * /LIMIT.SW2
                + /LIMIT.SW1 * LIMIT.SW2)
LIGHT = MASTER & MASTERSTOP
PANEL = MASTER & (RUN + START)
POWER = MASTER & /SAFETYVALVE
WHISTLE = MASTER & (SAFETY + STOP + /POWER)

```

In the above example, 19 instructions are saved by use of the X register.

In the event that some, but not all, of the controlling equations are controlled by some master condition, the X register must be loaded while the master equations are being evaluated, and must be "unloaded" or cleared when statements are encountered which do not utilize the X register feature. Clearing the X register actually means setting it to "1", and this is accomplished by loading the accumulator from address 0 (a guaranteed logic "1") and loading the X register from the accumulator. This operation is performed with the two instructions

```
LDA    0
```

```
AUX    776
```

### 2.3.3.2 The Y Register

A second optimization technique involves the use of the pseudo-jump feature generated by use of the Y register. Recall that setting the Y register to a logic "0" inhibits all STORE instructions. Using this feature one may simulate the results of a FORTRAN IF statement, and in a limited sense, the GO TO statement.

The use of the Y register is invoked by VIPTRAN's fifth binary operator, colon (:). Its use is analagous to that of the ampersand in that the colon operator loads the Y register with the content of the variable name immediately to its left if and only if that variable is not already stored in the Y register. Like "&", the priority of ":" is greater than the other binary operators (the priority of "&" and ":" is equal), so the loading of the Y register is the first code generated for an assignment statement using ":".

As an example, suppose the variable A should be set equal to  $X + Y * Z$  if B is true, and to  $X * Y + Z$  if B is false. The following three statements perform this operation:

$$(1) \text{ NOTB} = /B$$

$$(2) \text{ A} = \text{B} : \text{X} + /Y * Z$$

$$(3) \text{ A} = \text{NOTB} : \text{X} * /Y + Z$$

In the above case, only one of equations (2) and (3) can execute its STO instruction in any one memory scan. Note that equation (1) should precede the other two; if (1) and (2) were to be interchanged, the Y register would be loaded with B during execution of the first equation, unloaded (or cleared) during execution of  $\text{NOTB} = /B$ , and loaded with NOTB during execution of equation (3). Although this sequence remains logically correct, and codes correctly, it is needlessly inefficient.

### 2.3.3.3 X and Y Registers in Combination

The full power of the Y register is not apparent until we see that entire blocks of code may be optionally, or conditionally, executed. As before, if the conditional assignment is dependent upon an expression, rather than a variable, a simple assignment statement eliminates the problem.

Example:

```
START-UP = START-SWITCH * LINEPOWER * /RESET
RUN = /START-UP
A = START-UP : LIMIT.SW1 * LIMIT.SW2
B = START-UP : PANEL + MANUAL
C = START-UP : REMOTE + LOCAL
A = RUN : LIMIT.SW1 + LIMIT.SW2
B = RUN : PANEL * MANUAL
C = RUN : REMOTE + TEST
```

It is not unusual to find all the operators in use in a simple assignment statement.

```
MASTER1 = LINEPOWER * /STOP + MANUAL
MASTER2 = SAFETY + ABORT + OFF
A = MASTER1 & START-UP : LIMIT.SW1 * LIMIT.SW2
B = MASTER1 & START-UP : PANEL + MANUAL
C = MASTER2 & START-UP : REMOTE + LOCAL
A = MASTER1 & RUN : LIMIT.SW1 + LIMIT.SW2
B = MASTER1 & RUN : PANEL * MANUAL
C = MASTER2 & RUN : REMOTE + TEST
```

#### 2.3.3.4 The ON-OFF Variables

Occasionally the user simply wants to specifically set a variable (probably an output control) to a logic "1" (on) or to a logic "0" (off). For this reason two variable names are reserved which may be used as the Boolean constants. The variable names are:

"1"            Boolean "true"

"0"            Boolean "false"

Although these variables may be used in any expression, they find special use in conjunction with the Y register. For example, assume that MOTOR and LIGHT should be "on" if SWITCH is "on", but should retain their former state (whichever it is) if SWITCH is "off". This situation is expressed in VIPTRAN as:

MOTOR = SWITCH : "1"

LIGHT = SWITCH : "1"

The variable "0" is used analagously.

### 3. PROGRAM LOGIC MANUAL

#### 3.1 Overview of the Compiler's Responsibilities

The VIPTRAN compiler operates in a logical and predictable fashion. The purpose of this section is to explain exactly what internal logic decisions are made and what factors influence those choices. The compiler is comprised of 18 major sections, some dependent on and some independent of other sections. Most major sections have a number of subsections to perform repeated or specialized tasks. The major sections and their functions are:

<u>section name</u>	<u>function</u>
BATCH	the program batch driver; initializes the global variables for any "batch" of programs; sets parameters for compiler's table sizes and work space.
COMPILER	controls compilation phases for each individual program; performs symbol table and global variable initialization for each individual program; selects \$VIP card compiler options; controls optional invocation of all working procedures.
LEXI	lexical analysis; collects tokens from the source cards and classifies them semantically as identifiers, numbers, or operators.
LOOKUP	symbol table manager; finds and inserts identifiers and keywords in the symbol table.

<u>section name</u>	<u>function</u>
MAPPING	outputs a variable name and address map, sorted by ascending address.
ERROR	error message writer; prints six classes of error messages and sets severity code of error.
SYNA	syntactic analysis; examines the input stream and verifies each VIPTRAN statement for syntactic correctness, correcting errors whenever possible.
ADCHECK	checks programmer-defined variable addresses for proper upper and lower bounds, duplication, and type.
EXPRESSION	syntactic analysis and postfixing; examines each VIPTRAN assignment statement for syntactic correctness, correcting errors whenever possible; reduces the infix expression to Polish postfix; recognizes and translates timer functions.
OUT	manages postfix stack.
DUMP	prints compiler's internal variables, stacks, and tables for inspection by systems programmer.
SEMA	performs semantic typing of variables; determines type and address of undeclared variables.
CODEGEN	the code generator; translates Polish postfix from SYNA into VIP machine code.
TEMPGEN	allocates variable and address for compiler-defined temporary variables.
GEN	manages the code generator stack; inserts AUX instructions into output coding stream for

<u>section name</u>	<u>function</u>
	optimal (least number of instructions) scratchpad addressing.
OBJECT	formats, edits, and prints the final output page (object code) of VIP machine code.
PRINT	formats and prints the sorted list of source equations, and retrieves same from an indexed sequential direct access data set.
SORT	determines what interdependencies exist among the source equations and reorders them as necessary to prevent use of uninitialized variables.

These sections are logically connected as shown in Figure 2.

### 3.2 Module Technical Descriptions

The following section describes each of the 18 major modules in detail, with emphasis upon the logical sequence of operation of each.

#### 3.2.1 BATCH

A "job" consists of any number of sequential programs. BATCH separates the input stream into programs and monitors their compilation, and for each program initializes the global parameters affecting the compiler's internal size. These internal limits include:

- (1) the number of VIPIRAN assignment statements in the PROGRAM segment, currently 512;
- (2) the number of hash buckets for the symbol table, currently 257;
- (3) the length of the array containing the postfix representation of the source program, currently 3000 words;

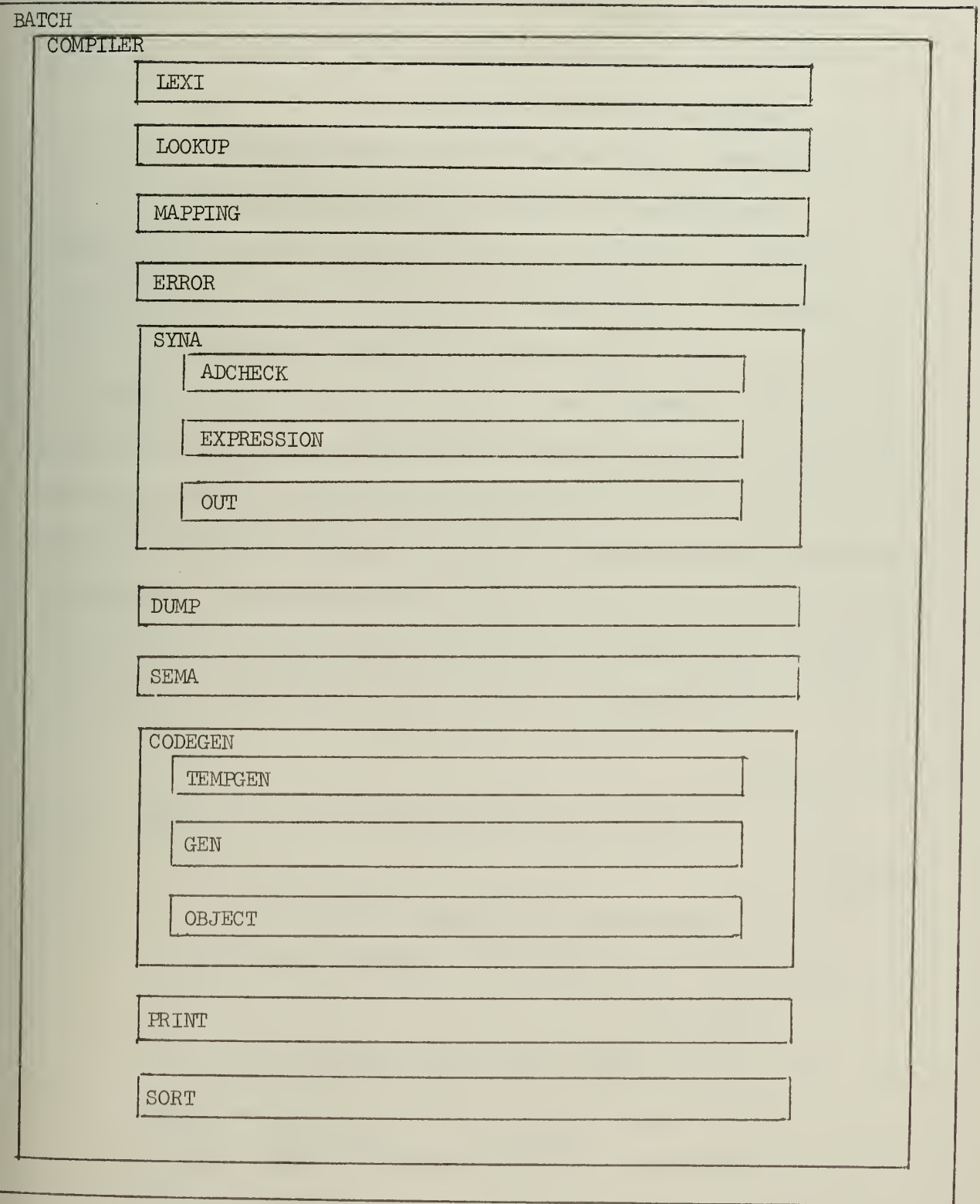


Figure 2. VIPTRAN Compiler Module Containment Map

- (4) size of the symbol table, representing the maximum number of user-defined and compiler-defined variables in any one program, currently 600;
- (5) number of reserved keywords recognized by the compiler and stored in the symbol table, currently 23.

Execution of the compiler begins with a scan of the input stream for a source card containing the four characters "\$VIP" in card columns 1-4. When such a card is located control passes to COMPILER. If no such card is found the compiler takes a "normal exit" by simply returning control to the operating system with no condition codes set. When a \$VIP card has been recognized and the program following it has been processed by COMPILER, control returns to BATCH and a search for another \$VIP card begins. Thus, BATCH terminates (1) if no \$VIP card occurs in the input stream, or (2) when control returns from COMPILER and no more \$VIP cards occur in the input stream.

### 3.2.2 COMPILER

This is the initialization and sequence of control module. COMPILER initializes:

- (1) an address map of 1024 bits;
- (2) flag bits for each of the compiler options on the \$VIP card;
- (3) the symbol table, each entry containing fields for the variable's character string name (12 or fewer characters), its octal address, its type, a map and count of all its uses in the program, and a hash collision resolution chain;

- (4) an order table which stores the position of the equation in its final (sorted) order, pointers to the head and tail of the postfix array section containing the translated equation, and the physical card number on which the source equation started (supplied by the compiler and used as a key for the disk copy of the input);
- (5) the hash table of prime length.

The time and date are then fetched from the system clock, the external file for source card images is created, the \$VIP card is examined for the nine compiler options which may be specified and appropriate flags set, the symbol table is initialized with 23 reserved keywords, the compiler heading lines (version number, time, date and options in use) are printed, and then begins a series of calls directed by the compiler's flag bits and the current error status of the program. The sequence is:

```

SYIA
DUMP
SEMA
DUMP
if MAP option enabled, then MAPPING
if terminal error has occurred, return to BATCH
if SORT option enabled, then SORT
DUMP
if a terminal error has occurred, return to BATCH
if PRINTSORT option enabled, then PRINT
DUMP
if CODE option enabled, then CODEGEN
DUMP
return to BATCH

```

The multiple calls to DUMP provide the systems programmer with an opportunity to examine the dynamic status of the compiler's stacks, tables, and internal variables, more thoroughly described in section DUMP.

### 3.2.3 LEXI

Lexi identifies the next token of the input stream, defined to be the longest contiguous string of a letter followed by zero or more letters and/or digits (an identifier or keyword), a digit followed by zero or more digits (an address), an operator (/, \*, +, &, :, =), or delimiter (left parenthesis, right parenthesis, comma, space, end-of-line marker, end-of-statement marker).

The space is the most common delimiter and is used to separate tokens. The character string X123 unambiguously represents the variable name X123, while X 123 is the declaration of the variable X and its assignment to address 123<sub>8</sub>.

Each source card is examined between columns 1-72 (only). The remaining columns may be used for any purpose, possibly sequence identification.

While the text of the declaration and program segments must be contained in columns 2-72, column 1 is always examined and the card processed according to its content:

<u>character in column 1</u>	<u>action</u>
C	This card is a comment; columns 1-72 of the card are printed in the source listing.
>	Used only in the program segment, it identifies the current card as a logical extension of the previous card (continuation card). Its use in the declaration segment has no effect.

character in column 1action

\$

\$ signifies a special control operation.

If columns 1-10 contain \$TRANSLATE, the syntax parse will end; if columns 1-5 contain \$STOP, the compiler halts; otherwise, columns 2 and 3 are taken as a two digit code for dumping the compiler's stacks and tables. The codes are explained in section DUMP.

blank

In the program segment, a blank in column 1 logically ends the previous assignment statement and restarts the assignment statement parser.

If the compiler is examining the program segment, a copy of the source cards is filed on disk to facilitate later printing of the sorted equations.

If, during the assignment statement parse, the input stream is exhausted and no \$TRANSLATE card has been found, a \$TRANSLATE card is inserted by the compiler, a non-terminal error message printed, and the assignment statement parse terminates normally.

Identification of a token containing more than 12 characters results in an error message; the first 12 characters of the token are used in place of the offending identifier.

If a token is an identifier, it is located or inserted in the symbol table and semantically classified as a variable or compiler keyword.

### 3.2.4 LOOKUP

The symbol table manager uses the method of hash buckets with direct chaining. The hash key is defined by a simple function

$$KEY = \left( \sum_{i=1}^L C_i 2^{L-i} \right) \bmod 257$$

where KEY is the hash key (hash bucket pointer),  $C_i$  is the 8-bit EBCDIC code for the  $i^{\text{th}}$  character in the token, and L is the number of characters in the token.

The function assures that every character in the token participates in the resultant key. The function is implemented by addition and shifting to achieve maximum speed. In the event of collisions, the chain field of the symbol table is a linked list which is sequentially searched. This lookup method is one of the fastest known but does require a moderate amount of storage for the hash buckets and chain field. Furthermore, the method allows unlimited growth of the symbol table and the number of hash buckets by merely changing the appropriate global variables in BATCH.

### 3.2.5 MAPPING

This module produces a listing of the program variables, sorted by ascending octal address, without actually sorting (moving) any data. The output lists the variable name, its address, and its type. If the variable name is a timer, its input is identified; if a timer function, the timer base variable and its input are identified. All of this information is recorded in the symbol table and the cross-references are obtained by following linked lists of maximum length two.

### 3.2.6 ERROR

The error message writer produces an English language error message and a compiler-generated card number or statement number for easy reference. The message is printed on the line below the offending statement if the error is syntactic, and on the line(s) below the \$TRANSLATE statement if semantic. Other error messages, such as SORT failure or code generator failure, also appear below the \$TRANSLATE statement. If appropriate, a portion of the source text is included in the error message to identify the left- or right-context of the error. A complete list of error messages, their causes, and some examples, are contained in Appendix C.

### 3.2.7 SYNA

The syntactic analysis of the source program is performed by SYNA. The declaration and program segments are examined separately. The declaration parse searches for the first occurrence of a keyword (INPUT6., INPUT12., INPUT24., INPUT48., INPUT120., OUTPUTAC., OUTPUTDC., TIMER., LATCH., or SCRATCH.) and then types all variables between the current and next keyword to be of the current type. Identification of the PROGRAM. keyword terminates the declaration parse and initiates the assignment statement parse. SYNA recognizes the target (left-hand side variable) of an assignment statement and the assignment operator (=) and calls EXPRESSION to translate the right-hand side expression into postfix. This operation is repeated for each assignment statement. Recognition of the \$TRANSLATE card terminates the assignment statement parse.

### 3.2.8 ADCHECK

The address checking module ascertains that no variable is multiply or ambiguously defined. Variable addresses are error checked for proper

bounds depending upon the type of variable, and assigned addresses are recorded in the symbol table.

### 3.2.9 EXPRESSION

VIPTRAN expressions are translated into postfix using operator precedence functions. EXPRESSION also translates the six timer functions directly into postfix, using the input variable cross-reference for timers available in the symbol table. An identifier will be interpreted as a timer function only if all of the following conditions are satisfied:

- (1) the token contains the character "-";
- (2) the "-" is between character positions 2 and 9, inclusive, of the token;
- (3) all characters to the right of "-" are either the letter O or the letter X;
- (4) the first three characters to the right of the "-" are one of the legal function codes: OOX, OXO, OXX, XOO, XOX, or XXO;
- (5) the character string to the left of "-" is a valid variable name which has been declared to be of type TIMER.

### 3.2.10 OUT

This simple module checks the postfix stack for overflow and either issues the appropriate stack overflow error message or adds the current token to the postfix stack.

### 3.2.11 DUMP

The successful development of a compiler is hastened by being able to examine the internal tables, stacks, pointers, and major variables

dynamically. Rather than insert and remove this code as needed, it has been permanently included as a separate module and is optionally invoked at any of seven distinct times in the compilation process. A simple coding scheme allows the systems programmer to cause the compiler to print its tables for examination. This option is invoked by including a source card of the form

$$\$ \text{digit}_1 \quad \text{digit}_2$$

using columns 1-3. Such a command card may appear anywhere in the source deck and does not disturb the normal operations of the compiler. The first digit specifies the time that the printout is to occur and the second digit specifies exactly what is to be printed. The possibilities are:

<u>digit<sub>1</sub></u>	<u>when DUMP is invoked</u>
0	immediately
1	after syntactic analysis
2	after semantic analysis
3	after sorting
4	after printing of sorted equations
5	after code has been generated

<u>digit<sub>2</sub></u>	<u>what is printed</u>
0	variable address allocation map; VIP card type allocation map
1	hash table
2	symbol table
3	postfix stack
4	sorted order table

Thus, the systems programmer could include cards such as

\$11

\$52

to examine the hash table after syntactic analysis and the symbol table after code generation.

### 3.2.12 SEMA

Semantic analysis determines the type and address of undeclared variables and the optimal distribution of VIP plug-in cards. The symbol table entries are scanned in three passes to accomplish optimal address assignment.

Pass 1. Although the lowest order plug-in card (addresses 0-17<sub>8</sub>) must be of type INPUT, it may equally well be of the 6, 12, 24, 48, or 120 volt variety. The first pass through the symbol table determines which input type (voltage) has been used, and in the rare event of no inputs assigns type INPUT120 to card position 1. The first pass is usually quite fast since inputs are traditionally declared first.

Pass 2. All user-variable addresses must be allocated before compiler-assigned addresses can be allocated. A second pass through the symbol table determines what card types are required by the programmer-supplied addresses and assigns VIP cards as required. At the same time all variables whose address map to the same card are error checked to assure that those variables are of identical type.

Pass 3. All variables which have not been assigned addresses, whether declared or not, are given physical locations by pass 3. If a variable is declared to be of type x, the current VIP cards are examined to find one of type x which is not full. If such a card is found, the first (lowest number) address is allocated. If all cards of type x are full, a new card

of type x is added to the card map and its first position is allocated. If a variable has not been declared, it may now be reasonably assumed to be a virtual control relay (i.e., a programmer defined variable with no connection to the outside world). If the SCRATCHPAD option is in use, the variable is assigned to an unfilled scratchpad location; if no scratchpad is allowed (the NOSCRATCHPAD option), the variable is allocated to the first unused position on the first partially filled OUTPUTAC or OUTPUTDC card, using the algorithm previously described.

Having assigned addresses to all variables, the VIP card requirements are now known, so the VIP card map is printed.

### 3.2.13 SORT

The SORT module determines which assignment statements use a variable on the right-hand (expression) side of the assignment operator when that same variable is also in use on the left-hand (target) side of some other equation(s). For all such situations the sorter attempts to place the definition (target side use) of the variable prior to all of the expression-side uses of that variable. The sorter uses a 512 by 512 bit matrix supplied by SYNA, and the count field for each variable in the symbol table, to speedily determine which variables, and hence which equations, must be considered. Sorting time is kept to a minimum by changing statement number indices in arrays, and by not moving characters strings themselves.

In the event that two or more equations are interlocked, the sorter drops the last equation of the set (the one with the largest VIPTRAN statement number), issues a warning message indicating which equation has been dropped, and proceeds to resort the remaining equations. The algorithm

necessarily terminates, since in the worst case of every equation being interlocked with every other equation, all but the first equation will eventually be dropped and the sort is "successful."

The logic of the sorter was designed and first implemented by Professor T. A. Murrell.

### 3.2.14 CODEGEN

CODEGEN is the master module for code generation. It always prints the page headings and the machine initialization sequence

LDA	0	load logic "1"
AUX	776	enable X register
AUX	775	enable Y register
AUX	774	enable scratchpad mode
STO	0	store "1" at 1000 <sub>8</sub>

Since each scan of the VIP's read-only memory executes these five instructions the X and Y registers, if altered by the execution of the previous scan, are now restored for proper execution of the current scan.

Each equation in sorted order is now examined sequentially (if the NOSORT option was in effect, sorted order is identically the original order). For each equation, the following operations are performed:

- (1) All compiler-generated temporary variables allocated by the previous equation are released by "unmarking" the address map:
- (2) The X and/or Y register might need to be reset if the previous equation used that feature and the current equation does not use that feature, or does not use it with the same variable. The compiler considers 5 cases to produce optimal code.

Case 1

The previous equation used both the X and Y registers; the current equation uses neither. Both registers must be cleared. The emitted code is:

```
LDA      0      load logic "1"
AUX      776     enable X register
AUX      775     enable Y register
```

Case 2

The previous equation used the X register; the current equation does not. The X register is cleared by emitting:

```
LDA      0      load logic "1"
AUX      776     enable X register
```

Case 3

The previous equation did not use the X register and the current equation does, or the previous equation used the X register and the current equation uses it with a different variable. The X register is loaded with the new variable by emitting:

```
LDA      <variable address>
AUX      776     load X register
```

Case 4

The previous equation used the Y register; the current equation does not. The Y register is cleared by emitting:

```
LDA      0      load logic "1"
AUX      775     enable Y register
```

Case 5

The previous equation did not use the Y register and the current equation does, or the previous equation used the Y register and the current equation uses it, but with a different variable. The Y register is cleared by emitting:

LDA      <variable address>

AUX      775      enable Y register

(3) Now begins the actual code generation for a given equation.

The compiler initializes a compile-time stack for use in code generation, utilizing the postfix form of the source equation. The standard techniques for translating postfix expressions are not directly applicable because of (1) the availability of AND-complement, OR-complement, and LDA-complement instructions, and (2) the emphasis on optimized code.

The coding loop will cause generation of a temporary storage location whenever necessary. The equation

$$X = (A + B) * (C + D)$$

whose postfix representation is (left to right)

$$X \ A \ B \ + \ C \ D \ + \ * \ =$$

is evaluated in the following steps.

<u>compile-time stack</u>	<u>code generator action</u>
X	
X A	
X A B	emit "LDA A"
X A B +	emit "OR B"
X t <sub>1</sub>	
X t <sub>1</sub> C	

compile-time stackcode generator action

X $t_1$ C D	generate temporary location TEMP to hold $t_1 = A + B$
X TEMP C	emit "STO TEMP"
X TEMP C D	emit "LDA C"
X TEMP C D +	emit "OR D"
X TEMP $t_2$	
X TEMP $t_2$ *	emit "AND TEMP"
X $t_3$	
X $t_3$ =	emit "STO X"

The lack of a store-complement instruction requires a complemented expression such as  $X = /(A+B)$  to be evaluated in the following steps:

LDA	A	load A
OR	B	OR with B
STO	TEMP	generate temporary storage space
LDAC	TEMP	load it back complemented
STO	X	final result

The compile-time stack is arbitrarily set to a maximum height of 20 (experience shows its average height is about 5). If a deeply nested equation attempts to overflow the stack, a terminal error is announced and code generation stops.

When all equations have been coded the code generator emits "AUX 771" which signifies the end of the program.

### 3.2.15 TEMPGEN

The compiler's temporary generator is called whenever the content of the accumulator must be retained for future use and yet the next instruction will be LDA or LDAC. The temporary location generated is chosen in accordance with this rule:

If the SCRATCHPAD option is in use, the first non-used scratchpad position is allocated; otherwise, the first non-used position on an already allocated (by SYNA) VIP plug-in card of type OUTPUTAC or OUTPUTDC is used. If, in the latter case, all VIP cards are full, a terminal error message is issued which explains the situation (lack of space) and suggests rerunning the program with the SCRATCHPAD option.

When a location is available for use, an entry is made in the symbol table using the variable name TEMPdddd, where dddd is the octal address of the temporary variable. The address is pushed onto a stack of code-generator defined temporaries and marked "not reusable" for the duration of the equation's translation.

The latter decision, while not strictly optimal since a given temporary location might be safely reused many times within the same equation, was made so that a field engineer could physically monitor all temporary results.

The stack of temporary variable addresses is arbitrarily set to a maximum height of 20. If an equation attempts to overflow the stack, a terminal error is issued and the code generator stops.

### 3.2.16 GEN

References to scratchpad addresses require adding instructions to the normal instruction stream to assure optimized code. A single reference to scratchpad (recognized by the address beginning  $\geq 1000_8$ ) should be preceded by the AUX 774 instruction. Sequential references to scratchpad of length 2 or more are better coded by considering the stream of scratchpad

referencing instructions to be a block, preceded by the AUX 773 instruction and followed by the AUX 772 instruction. By utilizing a two-level stack, GEN automatically inserts these AUX instructions into the output code emitted by CODEGEN.

### 3.2.17 OBJECT

This module is merely an editor for the final form of the object code. The VIP read-only memory word number is incremented by 1 for each successive instruction generated, except for the instruction following a STOR which is handled in accordance with the SKIPd option. The SKIPd option allows the programmer to automatically skip d memory locations after every STOR instruction. This feature can be used to introduce small "holes" in the read-only memory (treated as "no-operations" during VIP execution) for later adjustment or correction of code.

### 3.2.18 PRINT

The source equations are stored on disk as an indexed sequential data set and keyed by a four character code which represents the compiler-assigned card number assigned to that source image (and printed on the source listing). For speed, each source record contains a one character field which delimits the last card of a multi-card (continued) assignment statement. Thus, a request to print a particular assignment statement will actually print all the original source cards which were used to define that equation (no upper limit). This method of filing and retrieving the source data was tested and found to be significantly faster, more cost-effective, and less demanding of core storage than recreating the source text from the postfix.

#### 4. SUMMARY

The goal of VIPTRAN, like FORTRAN, is to save a programmer's time and reduce the frequency of coding errors by introducing the proper vehicle, in this case a new programming language, for the expression of algorithms.

The initial experience at Struthers-Dunn, using engineers with little or no programming experience, has been a definite decrease in both programming time and actual expense. Those persons using the language regard it favorably because VIPTRAN does allow one to think and design at a higher level than does an assembler language.

While the syntax of VIPTRAN was somewhat predicated on the knowledge of the VIP hardware, the techniques employed by the compiler are actually sufficiently general to allow conversion to a similar machine by changing only the structure of the code generator. This might well be an area for profitable future research.

It is anticipated that VIPTRAN, both language and compiler, will continue to change and improve in the future as they have in the past. Newer versions of the compiler will be released as necessary to keep the programming system viable.

I sincerely expect that future experiences with VIPTRAN will validate its primary goal: to make the human being more productive.

## LIST OF REFERENCES

- Gear, C. W., Computer Organization and Programming, McGraw-Hill, Inc., New York, 1969.
- Gries, D., Compiler Construction for Digital Computers, John Wiley and Sons, Inc., New York, 1971.
- Henry, Donald, ed., "Operating and Programming Manual," Struthers-Dunn, Bettendorf, Iowa, 1972.
- \_\_\_\_\_, "VIP Seminar Notes," Struthers-Dunn, Bettendorf, Iowa, 1972.
- Hopcroft, J. E. and J. D. Ullman, Formal Languages and their Relation to Automata, Addison-Wesley, Reading, Massachusetts, 1969.
- Kain, R. Y., Automata Theory: Machines and Languages, McGraw-Hill, Inc., New York, 1972.
- Knuth, D. E., The Art of Computer Programming, Addison-Wesley, Reading, Massachusetts, 1968.
- Wilcox, T. R., "Class notes for Computer Science 401 - Compiler Construction," unpublished, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1973.

## APPENDIX A

## VIPTRAN Programming Examples

VIPTRAN CCMPILER, VERSION 3.00  
 DATE: 09/03/73 TIME: 21.25.00.790

<OPTIONS> SOURCE NOSORT NOPRINTSORT CODE NOSORTFAIL MAP NOTTY NOSCRATCHPAD SKIPD

CARD#	STMT#	SOURCE STATEMENT
1	0	C
2	0	C PROGRAMMING EXAMPLE 1
3	0	C BANK VAULT BURGLAR ALARM, WITHOUT OPTIMIZATION
4	0	C
5	0	C PROGRAMMER: ALFRED C. WEAVER
6	0	C
7	0	C DECLARE INPUT VARIABLES AND THEIR ADDRESSES
8	0	C
9	0	C INPUT6. DOOR 1 SWITCH 2 VAULT 3 PUSHBUTTON 4
10	0	C
11	0	C DECLARE OUTPUT VARIABLES AND THEIR ADDRESSES
12	0	C
13	0	C OUTPUTDC. ALARM 20 TIMER 21
14	0	C
15	0	C BEGIN CONTROL PROGRAM SEGMENT
16	0	C
17	0	C PROGRAM.
18	1	C ALARM = DOOR*SWITCH + VAULT*SWITCH + PUSHBUTTON
19	2	C TIMER = /(DOOR*SWITCH + VAULT*SWITCH + PUSHBUTTON)
20	3	C \$TRANSLATE

## VIP CARD MAP

TYPE	ADDRESSES
INPUT6	0 -> 17
OUTPUTDC	20 -> 37

NAME	VARIABLE MAP ADDRESS TYPE
DOOR	1 INPUT6
SWITCH	2 INPUT6
VAULT	3 INPUT6
PUSHBUTTON	4 INPUT6
ALARM	20 OUTPUTDC
TIMER	21 OUTPUTDC

## VIP MACHINE CODE

WORD#	INSTR	ADDR	VARIABLE NAME
0000	LDA	000	
0001	AUX	776	
0002	AUX	775	
0003	AUX	774	
0004	STO	000	
0005	LDA	001	DOOR
0006	AND	002	SWITCH
0007	STO	022	TEMP0022
0010	LDA	003	VAULT
0011	AND	002	SWITCH
0012	OR	022	TEMP0022
0013	OR	004	PUSHBUTTON
0014	STO	020	ALARM
0015	LDA	001	DOOR
0016	AND	002	SWITCH
0017	STO	022	TEMP0022
0020	LDA	003	VAULT
0021	AND	002	SWITCH
0022	OR	022	TEMP0022
0023	OR	004	PUSHBUTTON
0024	STO	023	TEMP0023
0025	LDAC	023	TEMP0023
0026	STO	021	TIMER
0027	AUX	771	

VIPTRAN COMPILER, VERSION 3.00  
 DATE: 09/03/73 TIME: 21.25.03.080

<OPTIONS> SOURCE NOSORT NOPRINTSORT CODE NOSORTFAIL MAP NOTTY NOSCRATCHPAD SKIPO

CARD#	STMT#	SOURCE STATEMENT
1	0	C
2	0	C PROGRAMMING EXAMPLE 2
3	0	C BANK VAULT BURGLAR ALARM, WITH OPTIMIZATION
4	0	C
5	0	C PROGRAMMER: ALFRED C. WEAVER
6	0	C
7	0	C DECLARE INPUT VARIABLES AND THEIR ADDRESSES
8	0	C
9	0	INPUT6. DOOR 1 SWITCH 2 VAULT 3 PUSHBUTTON 4
10	0	C
11	0	C DECLARE OUTPUT VARIABLES AND THEIR ADDRESSES
12	0	C
13	0	OUTPUTDC. ALARM 20 TIMER 21
14	0	C
15	0	C BEGIN CONTROL PROGRAM SEGMENT
16	0	C
17	0	PROGRAM.
18	1	ALARM = (DOOR + VAULT) * SWITCH * PUSHBUTTON
19	2	TIMER = /ALARM
20	3	\$TRANSLATE

## VIP CARD MAP

TYPE	ADDRESSES
INPUT6	0 -> 17
OUTPUTDC	20 -> 37

VARIABLE MAP

NAME	ADDRESS	TYPE
------	---------	------

DOOR	1	INPUT6
SWITCH	2	INPUT6
VAULT	3	INPUT6
PUSHBUTTON	4	INPUT6
ALARM	20	OUTPUTDC
TIMER	21	OUTPUTDC

## VIP MACHINE CODE

WORD#	INSTR	ADDR	VARIABLE NAME
0000	LDA	000	
0001	AUX	776	
0002	AUX	775	
0003	AUX	774	
0004	STO	000	
0005	LDA	001	DOOR
0006	OR	003	VAULT
0007	AND	002	SWITCH
0010	AND	004	PUSHBUTTON
0011	STO	020	ALARM
0012	LDAC	020	ALARM
0013	STO	021	TIMER
0014	AUX	771	

VIPTRAN CCMPILER, VERSION 3.00  
DATE: 09/03/73 TIME: 21.25.05.140

<OPTIONS> SOURCE NOSORT NOPRINTSORT CODE NOSORTFAIL MAP NOTTY NOSCRATCHPAD SKIPO

CARD#	STMT#	SOURCE STATEMENT
1	0	C
2	0	C PROGRAMMING EXAMPLE 3
3	0	C AUTOMATIC WELDER CONTROLLER
4	0	C
5	0	C PROGRAMMER: ALFRED C. WEAVER
6	0	C
7	0	C DECLARE OUTPUT VARIABLES AND THEIR ADDRESSES
8	0	C
9	0	C OUTPUTAC. LEFT 20 RIGHT 21 UP 22 DOWN 23 WELD 24
10	0	C
11	0	C DECLARE INPUT VARIABLES
12	0	C LET COMPILER ASSIGN ADDRESSES TO INPUTS
13	0	C
14	0	C INPUT120. LS1 LS2 START
15	0	C
16	0	C DECLARE TIMER VARIABLES
17	0	C
18	0	C ONLY THE PROGRAMMER KNOWS THAT ONLY THE TWO TIMER FUNCTIONS
19	0	C -OOX AND -XXO, WHICH DO NOT REQUIRE KNOWLEDGE OF THE TIMER'S
20	0	C INPUT STATE, WILL BE USED IN THIS PROGRAM.
21	0	C
22	0	C TO GUARD AGAINST OMISSIONS, THE VIPTRAN LANGUAGE SYNTAX
23	0	C REQUIRES THE DECLARATION OF THE TIMER'S INPUT VARIABLE IN
24	0	C ALL CASES.
25	0	C
26	0	C THE LANGUAGE SYNTAX IS SATISFIED IN THIS EXAMPLE BY SUPPLYING
27	0	C A DUMMY VARIABLE, APPROPRIATELY NAMED 'DUMMY', AS THE TIMER'S
28	0	C INPUT VARIABLE. THIS DUMMY VARIABLE WILL NOT APPEAR IN THE
29	0	C OUTPUT CODE.
30	0	C
31	0	C LET THE COMPILER ASSIGN ADDRESSES FOR THE TIMERS
32	0	C
33	0	C TIMER. T10 (DUMMY) T15 (DUMMY)
34	0	C
35	0	C BEGIN CONTROL PROGRAM SEGMENT
36	0	C
37	0	C PROGRAM.
38	1	C LEFT = T10-OOX * T15-OOX * /LS1
39	2	C DOWN = T10-OOX * T15-OOX * /LS2
40	3	C T10 = START
41	4	C RIGHT = T10-OOX
42	5	C T15 = T10-OOX
43	6	C UP = T10-XXO * /RIGHT
44	7	C WELD = LEFT + RIGHT
45	8	C \$TRANSLATE

## VIP CARD MAP

TYPE	ADDRESSES
INPUT120	0 -> 17
OUTPUTAC	20 -> 37
TIMER	40 -> 57

VARIABLE MAP  
NAME ADDRESS TYPE

LS1	1	INPUT120	
LS2	2	INPUT120	
START	3	INPUT120	
LEFT	20	OUTPUTAC	
RIGHT	21	OUTPUTAC	
UP	22	OUTPUTAC	
DOWN	23	OUTPUTAC	
WELD	24	OUTPUTAC	
DUMMY	25	OUTPUTAC	
T10	40	TIMER: INPUT= DUMMY	
T15	41	TIMER: INPUT= DUMMY	
T15-OOX		FUNCTION: TIMER= T15 INPUT= DUMMY	
T10-XXO		FUNCTION: TIMER= T10 INPUT= DUMMY	

## VIP MACHINE CODE

WORD#	INSTR	ADDR	VARIABLE NAME
-------	-------	------	---------------

0000	LDA	000	
0001	AUX	776	
0002	AUX	775	
0003	AUX	774	
0004	STO	000	
0005	LDA	040	T10
0006	AND	041	T15
0007	ANDC	001	LS1
0010	STO	020	LEFT
0011	LDA	040	T10
0012	AND	041	T15
0013	ANDC	002	LS2
0014	STO	023	DOWN
0015	LDA	003	START
0016	STO	040	T10
0017	LDA	040	T10
0020	STO	021	RIGHT
0021	LDA	040	T10
0022	STO	041	T15
0023	LDAC	040	T10
0024	ANDC	021	RIGHT
0025	STO	022	UP
0026	LDA	020	LEFT
0027	OR	021	RIGHT
0030	STO	024	WELD
0031	AUX	771	

VIPTRAN CCMPILER, VERSION 3.00  
 DATE: 09/03/73 TIME: 21.25.07.580

<OPTIONS> SOURCE NOSORT NOPRINTSORT CODE NOSORTFAIL MAP NOTTY NOSCRATCHPAD SKIPO

CARD#	STMT#	SOURCE STATEMENT
1	0	C
2	0	C PROGRAMMING EXAMPLE 4
3	0	C CONTROL EQUATIONS WITH COMMON MASTER CONTROL EXPRESSIONS
4	0	C NO X REGISTER OPTIMIZATION
5	0	C
6	0	C PROGRAMMER: ALFRED C. WEAVER
7	0	C
8	0	C DECLARE THE INPUT VARIABLES
9	0	C LET THE COMPILER ASSIGN THEIR ADDRESSES
10	0	C
11	0	INPUT120. RUN START MANUAL SAFETY
12	0	C
13	0	C DECLARATION KEYWORDS CAN APPEAR REPEATEDLY, AND IN ANY ORDER
14	0	C
15	0	INPUT120. LINEPOWER
16	0	C
17	0	C DECLARE THE OUTPUT VARIABLES
18	0	C LET THE COMPILER ASSIGN THEIR ADDRESSES
19	0	C
20	0	OUTPUTAC. LIGHT GATE BELL
21	0	C
22	0	C CONTROL PROGRAM SEGMENT
23	0	C
24	0	PROGRAM.
25	1	LIGHT = RUN * /START * LINEPOWER
26	2	GATE = (RUN + MANUAL) * LINEPOWER
27	3	BELL = (/SAFETY + /GATE) * LINEPOWER
28	4	\$TRANSLATE

## VIP CARD MAP

TYPE	ADDRESSES	
INPUT120	0 ->	17
OUTPUTAC	20 ->	37

VARIABLE MAP  
NAME ADDRESS TYPE

RUN	1	INPUT120
START	2	INPUT120
MANUAL	3	INPUT120
SAFETY	4	INPUT120
LINEPOWER	5	INPUT120
LIGHT	20	OUTPUTAC
GATE	21	OUTPUTAC
BELL	22	OUTPUTAC

## VIP MACHINE CODE

WORD#	INSTR	ADDR	VARIABLE NAME
0000	LDA	000	
0001	AUX	776	
0002	AUX	775	
0003	AUX	774	
0004	STO	000	
0005	LDA	001	RUN
0006	ANDC	002	START
0007	AND	005	LINEPOWER
0010	STO	020	LIGHT
0011	LDA	001	RUN
0012	OR	003	MANUAL
0013	AND	005	LINEPOWER
0014	STO	021	GATE
0015	LDAC	004	SAFETY
0016	ORC	021	GATE
0017	AND	005	LINEPOWER
0020	STO	022	BELL
0021	AUX	771	

VIPTRAN COMPILER, VERSION 3.00  
 DATE: 09/03/73 TIME: 21.25.09.780

<OPTIONS> SOURCE NOSORT NOPRINTSORT CODE NOSORTFAIL MAP NOTTY NOSCRATCHPAD SKIPO

CARD#	STMT#	SOURCE STATEMENT
1	0	C
2	0	C PROGRAMMING EXAMPLE 5
3	0	C CONTROL EQUATIONS WITH COMMON MASTER CONTROL EXPRESSIONS,
4	0	C UTILIZING X REGISTER OPTIMIZATION
5	0	C
6	0	C PROGRAMMER: ALFRED C. WEAVER
7	0	C
8	0	C DECLARE THE INPUT VARIABLES
9	0	C LET THE COMPILER ASSIGN THEIR ADDRESSES
10	0	C
11	0	INPUT120. RUN START MANUAL SAFETY
12	0	C
13	0	C DECLARATIONS OF A BLOCK MAY EXTEND OVER AS MANY SOURCE CARDS
14	0	C AS NECESSARY.
15	0	C
16	0	C LINEPOWER
17	0	C
18	0	C DECLARE THE OUTPUT VARIABLES
19	0	C LET THE COMPILER ASSIGN THEIR ADDRESSES
20	0	C
21	0	OUTPUTAC. LIGHT GATE BELL
22	0	C
23	0	C CONTROL PROGRAM SEGMENT
24	0	C
25	0	PROGRAM.
26	1	LIGHT = LINEPOWER & RUN * /START
27	2	GATE = LINEPOWER & (RUN + MANUAL)
28	3	BELL = LINEPOWER & (/SAFETY + /GATE)
29	4	\$TRANSLATE

## VIP CARD MAP

TYPE	ADDRESSES
INPUT120	0 -> 17
OUTPUTAC	20 -> 37

VARIABLE MAP  
NAME ADDRESS TYPE

RUN	1	INPUT120
START	2	INPUT120
MANUAL	3	INPUT120
SAFETY	4	INPUT120
LINEPOWER	5	INPUT120
LIGHT	20	OUTPUTAC
GATE	21	OUTPUTAC
BELL	22	OUTPUTAC

## VIP MACHINE CODE

WORD#	INSTR	ADDR	VARIABLE NAME
0000	LDA	000	
0001	AUX	776	
0002	AUX	775	
0003	AUX	774	
0004	STO	000	
0005	LDA	005	LINEPOWER
0006	AUX	776	
0007	LDA	001	RUN
0010	ANDC	002	START
0011	STO	020	LIGHT
0012	LCA	001	RUN
0013	OR	003	MANUAL
0014	STO	021	GATE
0015	LDAC	004	SAFETY
0016	ORC	021	GATE
0017	STO	022	BELL
0020	AUX	771	

VIPTRAN COMPILER, VERSION 3.00  
 DATE: 09/03/73 TIME: 21.25.12.010

<OPTIONS> SOURCE NOSORT NOPRINTSORT CODE NOSORTFAIL MAP NOTTY NOSCRATCHPAD SKIPO

CARD#	STMT#	SOURCE STATEMENT
1	0	C
2	0	C PROGRAMMING EXAMPLE 6
3	0	C X REGISTER OPTIMIZATION OF A MASTER CONTROL EQUATION
4	0	C
5	0	C DECLARE 48-VOLT INPUTS
6	0	C
7	0	C PROGRAMMER: ALFRED C. WEAVER
8	0	C
9	0	C THE COMPILER WILL ASSIGN ALL VARIABLE ADDRESSES
10	0	C
11	0	C INPUT48. LIMIT.SW1 LIMIT.SW2
12	0	C
13	0	C DECLARE 6-VOLT INPUTS
14	0	C
15	0	C INPUT6. MASTERSTOP RUN START
16	0	C SAFETYVALVE POWER
17	0	C
18	0	C DECLARE DC OUTPUTS
19	0	C
20	0	C OUTPUTDC. LIGHT PANEL POWER WHISTLE
21	0	C
22	0	C CONTROL PROGRAM SEGMENT
23	0	C
24	0	C PROGRAM.
25	1	C
26	1	C THE FIRST TARGET VARIABLE, 'MASTER', IS AN VIRTUAL CONTROL
27	1	C RELAY, I.E., IT DOES NOT CONTROL AN EXTERNAL LOAD. BECAUSE
28	1	C IT IS NOT DECLARED THE COMPILER WILL ASSIGN IT TO AN UNUSED
29	1	C POSITION ON THE OUTPUTDC CARD.
30	1	C
31	1	C THE FIRST ASSIGNMENT STATEMENT DOES NOT FIT ON A SINGLE
32	1	C CARD, SO A CONTINUATION CARD IS EMPLOYED.
33	1	C
34	1	C MASTER = LINEPOWER * (LIMIT.SW1 * /LIMIT.SW2
35	1	C + /LIMIT.SW1 * LIMIT.SW2)
36	2	C
37	2	C LIGHT = MASTER & MASTERSTOP
38	3	C
39	3	C PANEL = MASTER & (RUN + START)
40	4	C
41	4	C POWER = MASTER & /SAFETYVALVE
42	5	C
43	5	C WHISTLE = MASTER & (SAFETY + STOP + /POWER)
44	6	C
45	6	C \$TRANSLATE

## VIP CARD MAP

TYPE	ADDRESSES
INPUT48	0 -> 17
INPUT6	20 -> 37
OUTPUTDC	40 -> 57
OUTPUTAC	60 -> 77

VARIABLE MAP  
NAME ADDRESS TYPE

LIMIT.SW1	1	INPUT48
LIMIT.SW2	2	INPUT48
MASTERSTOP	20	INPUT6
RUN	21	INPUT6
START	22	INPUT6
SAFETYVALVE	23	INPUT6
POWER	40	OUTPUTDC
LIGHT	41	OUTPUTDC
PANEL	42	OUTPUTDC
WHISTLE	43	OUTPUTDC
MASTER	60	OUTPUTAC
LINEPOWER	61	OUTPUTAC
SAFETY	62	OUTPUTAC
STOP	63	OUTPUTAC

## VIP MACHINE CODE

WORD#	INSTR	ADDR	VARIABLE NAME
0000	LDA	000	
0001	AUX	776	
0002	AUX	775	
0003	AUX	774	
0004	STO	000	
0005	LDA	001	LIMIT.SW1
0006	ANDC	002	LIMIT.SW2
0007	STO	044	TEMP0044
0010	LDAC	001	LIMIT.SW1
0011	AND	002	LIMIT.SW2
0012	OR	044	TEMP0044
0013	AND	061	LINEPOWER
0014	STO	060	MASTER
0015	LDA	060	MASTER
0016	AUX	776	
0017	LDA	020	MASTERSTOP
0020	STO	041	LIGHT
0021	LDA	021	RUN
0022	OR	022	START
0023	STO	042	PANEL
0024	LDAC	023	SAFETYVALVE
0025	STO	040	POWER
0026	LDA	062	SAFETY
0027	OR	063	STOP
0030	ORC	040	POWER
0031	STO	043	WHISTLE
0032	AUX	771	

VIPTRAN CCMPILER, VERSION 3.00  
 DATE: 09/03/73 TIME: 21.25.14.490

<OPTIONS> SOURCE NOSORT NOPRINTSORT CODE NOSORTFAIL MAP NOTTY NOSCRATCHPAD SKIPO

CARD#	STMT#	SOURCE STATEMENT
1	0	C
2	0	C PROGRAMMING EXAMPLE 7
3	0	C Y REGISTER OPTIMIZATION
4	0	C
5	0	C PROGRAMMER: ALFRED C. WEAVER
6	0	C
7	0	C THE COMPILER WILL ASSIGN ALL VARIABLE ADDRESSES
8	0	C
9	0	C DECLARE THE 24-VOLT INPUTS
10	0	C
11	0	C INPUT24. X Y Z
12	0	C
13	0	C DECLARE THE OUTPUT VARIABLE
14	0	C
15	0	C OUTPUTAC. A
16	0	C
17	0	C CONTROL PROGRAM SEGMENT
18	0	C
19	0	C PROGRAM.
20	1	C
21	1	C THE VARIABLE 'NOTB' HAS NOT BEEN DECLARED. IT WILL BE ASSIGNED
22	1	C TO AN UNUSED POSITION ON THE OUTPUTAC CARD.
23	1	C
24	1	C IN THE EQUATIONS BELOW, ONLY ONE OF EQUATIONS (2) AND (3)
25	1	C CAN EXECUTE ITS STO INSTRUCTION IN ANY ONE MEMORY SCAN.
26	1	C HENCE, 'A' CONDITIONALLY ASSUMES ONE OF TWO POSSIBLE DEFINITIONS
27	1	C DEPENDING UPON THE CURRENT VALUE OF THE INPUT 'B'.
28	1	C
29	1	C NOTB = / B
30	2	C A = B : X + /Y * Z
31	3	C A = NOTB : X * /Y + Z
32	4	C \$TRANSLATE

## VIP CARD MAP

TYPE	ADDRESSES
INPUT24	0 -> 17
OUTPUTAC	20 -> 37

## VARIABLE MAP

NAME	ADDRESS	TYPE
------	---------	------

X	1	INPUT24
Y	2	INPUT24
Z	3	INPUT24
A	20	OUTPUTAC
NOTB	21	OUTPUTAC
B	22	OUTPUTAC

## VIP MACHINE CODE

WORD#	INSTR	ADDR	VARIABLE NAME
0000	LDA	000	
0001	ALX	776	
0002	AUX	775	
0003	AUX	774	
0004	STO	000	
0005	LDAC	022	B
0006	STO	021	NOTB
0007	LCA	022	B
0010	AUX	775	
0011	LDAC	002	Y
0012	AND	003	Z
0013	OR	001	X
0014	STO	020	A
0015	LCA	021	NOTB
0016	AUX	775	
0017	LDA	001	X
0020	ANDC	002	Y
0021	OR	003	Z
0022	STO	020	A
0023	AUX	771	

VIPTRAN COMPILER, VERSION 3.00  
 DATE: 09/03/73 TIME: 21.25.16.730

<OPTIONS> SOURCE NOSORT NOPRINTSORT CODE NOSORTFAIL MAP NOTTY NOSCRATCHPAD SKIPO

CARD#	STMT#	SOURCE STATEMENT
1	0	C
2	0	C PROGRAMMING EXAMPLE 8
3	0	C Y REGISTER OPTIMIZATION - CONDITIONAL EXECUTION
4	0	C
5	0	C PROGRAMMER: ALFRED C. WEAVER
6	0	C
7	0	C THE COMPILER WILL ASSIGN ALL VARIABLE ADDRESSES
8	0	C
9	0	C DECLARE THE INPUTS
10	0	C
11	0	INPUT12. START-SWITCH LIMIT.SW1 LIMIT.SW2
12	0	INPUT120. LINEPOWER PANEL
13	0	INPUT24. RESET MANUAL REMOTE LOCAL TEST
14	0	C
15	0	C DECLARE THE OUTPUTS
16	0	C
17	0	OUTPUTDC. START-UP RUN A B C
18	0	C
19	0	C CONTROL PROGRAM SEGMENT
20	0	C
21	0	PROGRAM.
22	1	C
23	1	C 'START-UP' AND 'RUN' ARE THE Y REGISTER CONTROL VARIABLES
24	1	C
25	1	START-UP = START-SWITCH * LINEPOWER * /RESET
26	2	RUN = /START-UP
27	3	A = START-UP : LIMIT.SW1 * LIMIT.SW2
28	4	B = START-UP : PANEL + MANUAL
29	5	C = START-UP : REMOTE + LOCAL
30	6	A = RUN : LIMIT.SW1 + LIMIT.SW2
31	7	B = RUN : PANEL * MANUAL
32	8	C = RUN : REMOTE + TEST
33	9	\$TRANSLATE

## VIP CARD MAP

TYPE	ADDRESSES
INPUT12	0 -> 17
INPUT120	20 -> 37
INPUT24	40 -> 57
OUTPUTDC	60 -> 77

VARIABLE MAP  
NAME ADDRESS TYPE

START-SWITCH	1	INPUT12
LIMIT.SW1	2	INPUT12
LIMIT.SW2	3	INPUT12
LINEPOWER	20	INPUT120
PANEL	21	INPUT120
RESET	40	INPUT24
MANUAL	41	INPUT24
REMOTE	42	INPUT24
LOCAL	43	INPUT24
TEST	44	INPUT24
START-UP	60	OUTPUTDC
RUN	61	OUTPUTDC
A	62	OUTPUTDC
B	63	OUTPUTDC
C	64	OUTPUTDC

## VIP MACHINE CODE

WORD#	INSTR	ADDR	VARIABLE NAME
0000	LDA	000	
0001	AUX	776	
0002	AUX	775	
0003	AUX	774	
0004	STO	000	
0005	LDA	001	START-SWITCH
0006	AND	020	LINEPOWER
0007	ANDC	040	RESET
0010	STO	060	START-UP
0011	LDAC	060	START-UP
0012	STO	061	RUN
0013	LCA	060	START-UP
0014	AUX	775	
0015	LDA	002	LIMIT.SW1
0016	AND	003	LIMIT.SW2
0017	STO	062	A
0020	LDA	021	PANEL
0021	OR	041	MANUAL
0022	STO	063	B
0023	LDA	042	REMOTE
0024	OR	043	LOCAL
0025	STO	064	C
0026	LCA	061	RUN
0027	AUX	775	
0030	LDA	002	LIMIT.SW1
0031	OR	003	LIMIT.SW2
0032	STO	062	A
0033	LDA	021	PANEL
0034	AND	041	MANUAL
0035	STO	063	B
0036	LCA	042	REMOTE
0037	OR	044	TEST
0040	STO	064	C
0041	AUX	771	

VIPTRAN COMPILER, VERSION 3.00  
 DATE: 09/03/73 TIME: 21.25.19.260

<OPTIONS> SOURCE NOSORT NOPRINTSORT CODE NOSORTFAIL MAP NOTTY NOSCRATCHPAD SKIPO

CARD#	STMT#	SOURCE STATEMENT
1	0	C
2	0	C PROGRAMMING EXAMPLE 9
3	0	C X AND Y REGISTER OPTIMIZATION IN COMBINATION
4	0	C
5	0	C PROGRAMMER: ALFRED C. WEAVER
6	0	C
7	0	C THIS EXAMPLE ILLUSTRATES THE COMPLICATED LOADING AND UNLOADING
8	0	C OF THE X AND Y REGISTERS WHICH IS AUTOMATICALLY HANDLED BY THE
9	0	C COMPILER.
10	0	C
11	0	C THE COMPILER WILL ASSIGN ALL VARIABLE ADDRESSES
12	0	C
13	0	C DECLARE THE OUTPUTS
14	0	C
15	0	C OUTPUTAC. A B C
16	0	C
17	0	C DECLARE THE INPUTS
18	0	C
19	0	C INPUT120. LINEPOWER STOP MANUAL SAFETY ABORT
20	0	C OFF START-UP LIMIT.SW1 LIMIT.SW2
21	0	C PANEL REMOTE LOCAL RUN TEST
22	0	C
23	0	C CONTROL PROGRAM SEGMENT
24	0	C
25	0	C PROGRAM.
26	1	C
27	1	C MASTER1 = LINEPOWER * /STOP + MANUAL
28	2	C
29	2	C MASTER2 = SAFETY + ABORT + OFF
30	3	C
31	3	C A = MASTER1 & START-UP : LIMIT.SW1 * LIMIT.SW2
32	4	C
33	4	C B = MASTER1 & START-UP : PANEL + MANUAL
34	5	C
35	5	C C = MASTER2 & START-UP : REMOTE + LOCAL
36	6	C
37	6	C A = MASTER1 & RUN : LIMIT.SW1 + LIMIT.SW2
38	7	C
39	7	C B = MASTER1 & RUN : PANEL * MANUAL
40	8	C
41	8	C C = MASTER2 & RUN : REMOTE + TEST
42	9	C
43	9	C \$TRANSLATE

## VIP CARD MAP

TYPE	ADDRESSES
------	-----------

INPUT120	0 -> 17
----------	---------

OUTPUTAC	20 -> 37
----------	----------

VARIABLE MAP  
NAME ADDRESS TYPE

LINEPOWER	1	INPUT120
STOP	2	INPUT120
MANUAL	3	INPUT120
SAFETY	4	INPUT120
ABORT	5	INPUT120
OFF	6	INPUT120
START-UP	7	INPUT120
LIMIT.SW1	10	INPUT120
LIMIT.SW2	11	INPUT120
PANEL	12	INPUT120
REMOTE	13	INPUT120
LOCAL	14	INPUT120
RUN	15	INPUT120
TEST	16	INPUT120
A	20	OUTPUTAC
B	21	OUTPUTAC
C	22	OUTPUTAC
MASTER1	23	OUTPUTAC
MASTER2	24	OUTPUTAC

## VIP MACHINE CODE

WORD#	INSTR	ADDR	VARIABLE NAME
0000	LDA	000	
0001	AUX	776	
0002	AUX	775	
0003	AUX	774	
0004	STO	000	
0005	LDA	001	LINEPOWER
0006	ANDC	002	STOP
0007	OR	003	MANUAL
0010	STO	023	MASTER1
0011	LDA	004	SAFETY
0012	OR	005	ABORT
0013	OR	006	OFF
0014	STO	024	MASTER2
0015	LDA	023	MASTER1
0016	AUX	776	
0017	LDA	007	START-UP
0020	AUX	775	
0021	LDA	010	LIMIT.SW1
0022	AND	011	LIMIT.SW2
0023	STO	020	A
0024	LDA	012	PANEL
0025	OR	003	MANUAL
0026	STO	021	B
0027	LDA	024	MASTER2
0030	AUX	776	
0031	LDA	013	REMOTE
0032	OR	014	LOCAL
0033	STO	022	C
0034	LDA	023	MASTER1
0035	AUX	776	
0036	LDA	015	RUN
0037	AUX	775	
0040	LDA	010	LIMIT.SW1
0041	OR	011	LIMIT.SW2
0042	STO	020	A
0043	LDA	012	PANEL
0044	AND	003	MANUAL
0045	STO	021	B
0046	LDA	024	MASTER2
0047	AUX	776	
0050	LDA	013	REMOTE
0051	OR	016	TEST
0052	STO	022	C
0053	AUX	771	

VIPTRAN COMPILER, VERSION 3.00  
 DATE: 09/03/73 TIME: 21.25.22.140

<OPTIONS> SOURCE SORT PRINTSORT CODE NOSORTFAIL MAP NOTTY NOSCRATCHPAD SKIPO

CARD#	STMT#	SOURCE STATEMENT
1	0	C
2	0	C PROGRAMMING EXAMPLE 10
3	0	C A PROGRAM TO ILLUSTRATE A SUCCESSFUL SORT OF INTERDEPENDENT
4	0	C ASSIGNMENT STATEMENTS. THE 'SORT' AND 'PRINTSORT'
5	0	C OPTIONS ARE ON.
6	0	C
7	0	C PROGRAMMER: ALFRED C. WEAVER
8	0	C
9	0	C DECLARE THE INPUTS
10	0	C
11	0	C INPUT24. SWITCH1 SWITCH2
12	0	C
13	0	C DECLARE THE OUTPUTS
14	0	C
15	0	C OUTPUTDC. A B C D E F
16	0	C HORN BELL
17	0	C
18	0	C PROGRAM.
19	1	C
20	1	C THE FOLLOWING SIX EQUATIONS ARE NOT IN OPTIMAL ORDER
21	1	C
22	1	C $A = (B+C) * (D+E)$
23	2	C $R = E+F$
24	3	C $C = B*C$
25	4	C $D = /(E+F)$
26	5	C $E = /E$
27	6	C $F = SWITCH1 + SWITCH2$
28	7	C
29	7	C \$TRANSLATE

VIP CARD MAP		
TYPE	ADDRESSES	
INPUT24	0 ->	17
OUTPUTDC	20 ->	37

VARIABLE MAP  
NAME ADDRESS TYPE

SWITCH1	1	INPUT24
SWITCH2	2	INPUT24
A	20	OUTPUTDC
B	21	OUTPUTDC
C	22	OUTPUTDC
D	23	OUTPUTDC
E	24	OUTPUTDC
F	25	OUTPUTDC
HORN	26	OUTPUTDC
BELL	27	OUTPUTDC

SOURCE EQUATIONS IN SORTED ORDER  
STMT#     SOURCE STATEMENT

1	$E = 1/E$
2	$F = \text{SWITCH1} + \text{SWITCH2}$
3	$C = B * C$
4	$D = 1/(E + F)$
5	$A = (B + C) * (D + E)$
6	$B = E + F$

## VIP MACHINE CODE

WORD#	INSTR	ADDR	VARIABLE NAME
0000	LDA	000	
0001	AUX	776	
0002	AUX	775	
0003	AUX	774	
0004	STO	000	
0005	LDAC	024	E
0006	STO	024	E
0007	LDA	001	SWITCH1
0010	OR	002	SWITCH2
0011	STO	025	F
0012	LDA	021	B
0013	AND	022	C
0014	STO	022	C
0015	LDA	024	E
0016	OR	025	F
0017	STO	030	TEMP0030
0020	LDAC	030	TEMP0030
0021	STO	023	D
0022	LDA	021	B
0023	OR	022	C
0024	STO	030	TEMP0030
0025	LDA	023	D
0026	OR	024	E
0027	AND	030	TEMP0030
0030	STO	020	A
0031	LDA	024	E
0032	OR	025	F
0033	STO	021	B
0034	AUX	771	

VIPTAN COMPILER, VERSION 3.00  
 DATE: 09/03/73 TIME: 21.25.26.930

<OPTIONS> SOURCE SORT PRINTSORT NOCODE NOSORTFAIL MAP NOTTY NOSCRATCHPAD SKIPO

CARD#	STMT#	SOURCE STATEMENT
1	0	C
2	0	C PROGRAMMING EXAMPLE 11
3	0	C A PROGRAM TO ILLUSTRATE AN UNSUCCESSFUL SORT
4	0	C
5	0	C THE 'SORT' AND 'PRINTSORT' OPTIONS ARE ON.
6	0	C
7	0	C PROGRAMMER: ALFRED C. WEAVER
8	0	C
9	0	C DECLARE THE INPUTS
10	0	C
11	0	C INPUT48. SWITCH1 SWITCH2
12	0	C
13	0	C DECLARE THE OUTPUTS
14	0	C
15	0	C OUTPUTAC. A B C D E F
16	0	C
17	0	C PROGRAM.
18	1	C
19	1	C THIS BLOCK OF EQUATIONS CAN NOT BE SATISFACTORILY REORDERED
20	1	C DUE TO INTERLOCKING
21	1	C
22	1	C $A = (B * C * D) + E$
23	2	C $B = (A + C + D) + E$
24	3	C $C = (A * B + D) + E$
25	4	C $D = (A + B * C) + E$
26	5	C $E = /E + F$
27	6	C $F = SWITCH1 * SWITCH2$
28	7	C
29	7	C \$TRANSLATE

VIP CARD MAP  
TYPE ADDRESSES  
INPUT48 0 -> 17  
OUTPUTAC 20 -> 37

## VARIABLE MAP

NAME	ADDRESS	TYPE
------	---------	------

SWITCH1	1	INPUT48
SWITCH2	2	INPUT48
A	20	OUTPUTAC
B	21	OUTPUTAC
C	22	OUTPUTAC
D	23	OUTPUTAC
E	24	OUTPUTAC
F	25	OUTPUTAC

SOURCE EQUATIONS IN SORTED ORDER  
STMT# SOURCE STATEMENT

```
1      B = (A + C + D) + E
2      D = (A + B * C) + E
3      C = (A * B + D) + E
4      A = (B * C * D) + E
5      E = /E + F
6      F = SWITCH1 * SWITCH2
```

VIPTRAN COMPILER, VERSION 3.00  
 DATE: 09/03/73 TIME: 21.25.31.240

OPTIONS> SOURCE NOSORT NOPRINTSORT CODE NOSORTFAIL MAP NOTTY SCRATCHPAD SKIPO

CARD#	STMT#	SOURCE STATEMENT
1	0	C
2	0	C PROGRAMMING EXAMPLE 12
3	0	C A PROGRAM TO ILLUSTRATE AUTOMATIC SCRATCHPAD ADDRESSING
4	0	C
5	0	C THE 'SCRATCHPAD' OPTION IS ON
6	0	C
7	0	C PROGRAMMER: ALFRED C. WEAVER
8	0	C
9	0	C DECLARE THE OUTPUTS
10	0	C
11	0	OUTPUTAC. CR3 CR6
12	0	C
13	0	C DECLARE THE SCRATCH VARIABLES
14	0	C
15	0	SCRATCH. CR1 CR2 CR4 CR5
16	0	C
17	0	C DECLARE THE INPUTS
18	0	C
19	0	INPUT120. SWITCH1 SWITCH2
20	0	LIGHT1 LIGHT2
21	0	BUTTON1 BUTTON2 BUTTON3
22	0	SAFETY1 SAFETY2 SAFETY3
23	0	C
24	0	PROGRAM.
25	1	C
26	1	CR1 = SWITCH1 + SWITCH2
27	2	CR2 = LIGHT1 + LIGHT2
28	3	CR3 = CR1 * /CR2 + /CR1 * CR2
29	4	CR4 = BUTTON1 * BUTTON2 * BUTTON3
30	5	CR5 = SAFETY1 + SAFETY2 + SAFETY3
31	6	CR6 = CR4 * /CR5 + /CR4 * CR5
32	7	C
33	7	\$TRANSLATE

VIP CARD MAP  
TYPE ADDRESSES  
INPUT120 0 -> 17  
OUTPUTAC 20 -> 37  
SCRATCHPAD 1000 -> 1777

VARIABLE MAP		
NAME	ADDRESS	TYPE
SWITCH1	1	INPUT120
SWITCH2	2	INPUT120
LIGHT1	3	INPUT120
LIGHT2	4	INPUT120
BUTTON1	5	INPUT120
BUTTON2	6	INPUT120
BUTTON3	7	INPUT120
SAFETY1	10	INPUT120
SAFETY2	11	INPUT120
SAFETY3	12	INPUT120
CR3	20	OUTPUTAC
CR6	21	OUTPUTAC
CR1	1001	SCRATCH
CR2	1002	SCRATCH
CR4	1003	SCRATCH
CR5	1004	SCRATCH

## VIP MACHINE CODE

WORD#	INSTR	ADDR	VARIABLE NAME
0000	LDA	000	
0001	AUX	776	
0002	AUX	775	
0003	AUX	774	
0004	STO	000	
0005	LDA	001	SWITCH1
0006	OR	002	SWITCH2
0007	AUX	774	
0010	STO	001	CR1
0011	LDA	003	LIGHT1
0012	OR	004	LIGHT2
0013	AUX	773	
0014	STO	002	CR2
0015	LDA	001	CR1
0016	ANDC	002	CR2
0017	STO	005	TEMP1005
0020	LDAC	001	CR1
0021	AND	002	CR2
0022	OR	005	TEMP1005
0023	AUX	772	
0024	STO	020	CR3
0025	LDA	005	BUTTON1
0026	AND	006	BUTTON2
0027	AND	007	BUTTON3
0030	AUX	774	
0031	STO	003	CR4
0032	LDA	010	SAFETY1
0033	OR	011	SAFETY2
0034	OR	012	SAFETY3
0035	AUX	773	
0036	STO	004	CR5
0037	LDA	003	CR4
0040	ANDC	004	CR5
0041	STO	005	TEMP1005
0042	LDAC	003	CR4
0043	AND	004	CR5
0044	OR	005	TEMP1005
0045	AUX	772	
0046	STO	021	CR6
0047	AUX	771	

VIPTRAN COMPILER, VERSION 3.00  
 DATE: 09/03/73 TIME: 21.25.33.440

<OPTIONS> SOURCE NOSORT NOPRINTSORT CODE NOSORTFAIL MAP NOTTY NOSCRATCHPAD SKIP3

CARD#	STMT#	SOURCE STATEMENT
1	0	C
2	0	C PROGRAMMING EXAMPLE 13
3	0	C A FULL SCALE EXAMPLE FOR A REAL SITUATION
4	0	C
5	0	C CONTROL PROGRAMMING FOR BAKER PLASTICS CORPORATION
6	0	C
7	0	C THE 'SKIP3' OPTION IS IN USE
8	0	C
9	0	C PROGRAMMER: ALFRED C. WEAVER
10	0	C
11	0	C DECLARE THE INPUTS
12	0	C
13	0	INPUT24. SS1 PL PB13 SS6 PB1 LS3 LS11
14	0	SS8A LS6 PB18 LS4 LS7 LS8 LS9
15	0	LS10
16	0	C
17	0	INPUT12. LS5
18	0	C
19	0	C DECLARE THE OUTPUTS AND THEIR ADDRESSES
20	0	C
21	0	OUTPUTDC. R22 755 R12 756 R15 757 R13 760
22	0	R11 761 R18 762 R10 763 R9 764 R8 765
23	0	R7 766 R14 767 R5 770 R4 771 R3 772 R20 773
24	0	R2 774 R16 775 R1 776 R21 777
25	0	C
26	0	C DECLARE LATCHING RELAYS
27	0	C
28	0	LATCH. T1IN T2IN 63 TR4INPUT TR7INPUT TR10INPUT
29	0	C
30	0	C DECLARE TIMERS
31	0	C
32	0	TIMER. T1 44 (T1IN 62)
33	0	T2 50 (T2IN)
34	0	TR8 (DUMMY)
35	0	TR2 (DUMMY)
36	0	TR3 (DUMMY)
37	0	TR9 (DUMMY)
38	0	TR7 (DUMMY)
39	0	TR10 (DUMMY)
40	0	TR6 (DUMMY)
41	0	C
42	0	PROGRAM.
43	1	C
44	1	T1IN=R21
45	2	T2IN = R21
46	3	R21 = (TR8-DOX+R21)*SS1*PL
47	4	R1=((1/R16+/R21)*R1+/SS1+PB13)*PL
48	5	MS=(/SS6*PB14+MS)*R1*PL
49	6	R2=SS6*R1*PL
50	7	TR2=(R6+R7)*R20*/R4*SS1*R1*PL
51	8	CC=(R3+TR2-DOX)*T2-XXO*SS1*R1*PL
52	9	R3=/R21*CC

53	10	TR8=/R20*CC
54	11	R4=LS3*R1*PL
55	12	R5=(R3*R4+R5)*R14*SS1*R1*PL
56	13	R6=LS11*R1*PL
57	14	R7=/LS11*R1*PL
58	15	R8=(R8+/R6)*R9*(R5*SS1+SS8A*/SS1)*R1*PL
59	16	R9=(R9+/R7)*R8*(R5*SS1+SS8A*/SS1)*R1*PL
60	17	R10=(R10+R18)*(R6*R8+R7*R9)*SS1*R1*PL
61	18	R11=(R5*R13*(R6*R8*R7*R9)+R11*/R15)*SS1*R1*PL
62	19	R12=TR3-OOX*/R14*R11
63	20	R13=LS6*R1*PL
64	21	TR4INPUT = (R11*R12+/R20*T1-OOX)*SS1*R1*PL
65	22	TR4 = TR4INPUT
66	23	TR10INP = TR9-OOX * TR4INPUT
67	24	TR10 = TR10INP
68	25	R14 = T1-XOX * TR4INPUT
69	26	TR7INPUT = T2-XOX * R14
70	27	TR7 = TR7INPUT
71	28	R15 = /R18*TR7-OOX*TR7INPUT
72	29	R22 = (PB18*/SS1+TR10-XXO*TR10INP)*R1*PL
73	30	R16=LS4*R1*PL
74	31	R17=(R15*R16*/R3*R17)*SS1*R1*PL
75	32	R18=LS7*LS8*LS9*LS10*R1*PL
76	33	R19=/R20*R17*TR6-OOX*R18
77	34	R20=LS5*R1*PL
78	35	\$TRANSLATE

## VIP CARD MAP

TYPE	ADDRESSES
INPUT24	0 -> 17
INPUT12	20 -> 37
TIMER	40 -> 57
LATCH	60 -> 77
OUTPUTAC	100 -> 117
OUTPUTDC	740 -> 757
OUTPUTDC	760 -> 777

VARIABLE MAP  
NAME ADDRESS TYPE

SS1	1	INPUT24
PL	2	INPUT24
PB13	3	INPUT24
SS6	4	INPUT24
PB1	5	INPUT24
LS3	6	INPUT24
LS11	7	INPUT24
SS8A	10	INPUT24
LS6	11	INPUT24
PB18	12	INPUT24
LS4	13	INPUT24
LS7	14	INPUT24
LS8	15	INPUT24
LS9	16	INPUT24
LS10	17	INPUT24
LS5	20	INPUT12
TR8	40	TIMER: INPUT= DUMMY
TR2	41	TIMER: INPUT= DUMMY
TR3	42	TIMER: INPUT= DUMMY
TR9	43	TIMER: INPUT= DUMMY
T1	44	TIMER: INPUT= T1 IN
TR7	45	TIMER: INPUT= DUMMY
TR10	46	TIMER: INPUT= DUMMY
TR6	47	TIMER: INPUT= DUMMY
T2	50	TIMER: INPUT= T2 IN
TR4INPUT	60	LATCH
TR7INPUT	61	LATCH
T1IN	62	LATCH
T2IN	63	LATCH
TR10INPUT	64	LATCH
DUMMY	100	OUTPUTAC
MS	101	OUTPUTAC
PB14	102	OUTPUTAC
R6	103	OUTPUTAC
CC	104	OUTPUTAC
TR4	105	OUTPUTAC
TR10INP	106	OUTPUTAC
R17	107	OUTPUTAC
R19	110	OUTPUTAC
R22	755	OUTPUTDC
R12	756	OUTPUTDC
R15	757	OUTPUTDC
R13	760	OUTPUTDC
R11	761	OUTPUTDC
R18	762	OUTPUTDC
R10	763	OUTPUTDC
R9	764	OUTPUTDC
R8	765	OUTPUTDC
R7	766	OUTPUTDC
R14	767	OUTPUTDC
R5	770	OUTPUTDC
R4	771	OUTPUTDC
R3	772	OUTPUTDC
R20	773	OUTPUTDC
R2	774	OUTPUTDC
R16	775	OUTPUTDC

R1	776	OUTPUTDC	
R21	777	OUTPUTDC	
TR6-00X	FUNCTION:	TIMER=	TR6 INPUT= DUMMY
TR10-XXO	FUNCTION:	TIMER=	TR10 INPUT= DUMMY
TR3-00X	FUNCTION:	TIMER=	TR3 INPUT= DUMMY
TR2-00X	FUNCTION:	TIMER=	TR2 INPUT= DUMMY
TR8-00X	FUNCTION:	TIMER=	TR8 INPUT= DUMMY
T2-XOX	FUNCTION:	TIMER=	T2 INPUT= T2IN
T1-XCX	FUNCTION:	TIMER=	T1 INPUT= T1IN

## VIP MACHINE CODE

WORD#	INSTR	ADDR	VARIABLE NAME
0000	LDA	000	
0001	AUX	776	
0002	ALX	775	
0003	AUX	774	
0004	STO	000	
0010	LDA	777	R21
0011	STO	062	T1IN
0015	LDA	777	R21
0016	STO	063	T2IN
0022	LCA	040	TR8
0023	OR	777	R21
0024	AND	001	SS1
0025	AND	002	PL
0026	STO	777	R21
0032	LCAC	775	R16
0033	ORC	777	R21
0034	AND	776	R1
0035	ORC	001	SS1
0036	OR	003	PB13
0037	AND	002	PL
0040	STO	776	R1
0044	LDAC	004	SS6
0045	AND	102	PB14
0046	OR	101	MS
0047	AND	776	R1
0050	AND	002	PL
0051	STO	101	MS
0055	LDA	004	SS6
0056	AND	776	R1
0057	AND	002	PL
0060	STO	774	R2
0064	LDA	103	R6
0065	OR	766	R7
0066	AND	773	R20
0067	ANDC	771	R4
0070	AND	001	SS1
0071	AND	776	R1
0072	AND	002	PL
0073	STO	041	TR2
0077	LDA	772	R3
0100	OR	041	TR2
0101	ANDC	050	T2
0102	AND	001	SS1
0103	AND	776	R1
0104	AND	002	PL
0105	STO	104	CC
0111	LDAC	777	R21
0112	AND	104	CC
0113	STO	772	R3
0117	LDAC	773	R20
0120	AND	104	CC
0121	STO	040	TR8
0125	LDA	006	LS3
0126	AND	776	R1
0127	AND	002	PL

0130	STO	771	R4
0134	LDA	772	R3
0135	AND	771	R4
0136	OR	770	R5
0137	ANDC	767	R14
0140	AND	001	SS1
0141	AND	776	R1
0142	AND	002	PL
0143	STO	770	R5
0147	LDA	007	LS11
0150	AND	776	R1
0151	AND	002	PL
0152	STO	103	R6
0156	LDAC	007	LS11
0157	AND	776	R1
0160	AND	002	PL
0161	STO	766	R7
0165	LDA	765	R8
0166	ORC	103	R6
0167	ANDC	764	R9
0170	STO	111	TEMP0111
0174	LDA	770	R5
0175	AND	001	SS1
0176	STO	112	TEMP0112
0202	LCA	010	SS8A
0203	ANDC	001	SS1
0204	OR	112	TEMP0112
0205	AND	111	TEMP0111
0206	AND	776	R1
0207	AND	002	PL
0210	STO	765	R8
0214	LCA	764	R9
0215	ORC	766	R7
0216	ANDC	765	R8
0217	STO	111	TEMP0111
0223	LCA	770	R5
0224	AND	001	SS1
0225	STO	112	TEMP0112
0231	LDA	010	SS8A
0232	ANDC	001	SS1
0233	OR	112	TEMP0112
0234	AND	111	TEMP0111
0235	AND	776	R1
0236	AND	002	PL
0237	STO	764	R9
0243	LCA	763	R10
0244	OR	762	R18
0245	STO	111	TEMP0111
0251	LCA	103	R6
0252	AND	765	R8
0253	STO	112	TEMP0112
0257	LDA	766	R7
0260	AND	764	R9
0261	OR	112	TEMP0112
0262	AND	111	TEMP0111
0263	AND	001	SS1
0264	AND	776	R1
0265	AND	002	PL
0266	STO	763	R10
0272	LDA	770	R5

0273	AND	760	R13
0274	STO	111	TEMP0111
0300	LDA	103	R6
0301	AND	765	R8
0302	AND	766	R7
0303	AND	764	R9
0304	AND	111	TEMP0111
0305	STO	112	TEMP0112
0311	LDA	761	R11
0312	ANDC	757	R15
0313	OR	112	TEMP0112
0314	AND	001	SS1
0315	AND	776	R1
0316	AND	002	PL
0317	STO	761	R11
0323	LDA	042	TR3
0324	ANDC	767	R14
0325	AND	761	R11
0326	STO	756	R12
0332	LDA	011	LS6
0333	AND	776	R1
0334	AND	002	PL
0335	STO	760	R13
0341	LDA	761	R11
0342	AND	756	R12
0343	STO	111	TEMP0111
0347	LDAC	773	R20
0350	AND	062	T1IN
0351	OR	111	TEMP0111
0352	AND	001	SS1
0353	AND	776	R1
0354	AND	002	PL
0355	STO	060	TR4INPUT
0361	LDA	060	TR4INPUT
0362	STO	105	TR4
0366	LDA	043	TR9
0367	AND	060	TR4INPUT
0370	STO	106	TR10INP
0374	LDA	106	TR10INP
0375	STO	046	TR10
0401	LDAC	062	T1IN
0402	OR	044	T1
0403	AND	060	TR4INPUT
0404	STO	767	R14
0410	LDAC	063	T2IN
0411	OR	050	T2
0412	AND	767	R14
0413	STO	061	TR7INPUT
0417	LDA	061	TR7INPUT
0420	STO	045	TR7
0424	LDAC	762	R18
0425	AND	045	TR7
0426	AND	061	TR7INPUT
0427	STO	757	R15
0433	LDA	012	PB18
0434	ANDC	001	SS1
0435	STO	111	TEMP0111
0441	LDAC	046	TR10
0442	AND	106	TR10INP
0443	OR	111	TEMP0111

0444	AND	776	R1
0445	AND	002	PL
0446	STO	755	R22
0452	LDA	013	LS4
0453	AND	776	R1
0454	AND	002	PL
0455	STO	775	R16
0461	LDA	757	R15
0462	AND	775	R16
0463	ANDC	772	R3
0464	AND	107	R17
0465	AND	001	SS1
0466	AND	776	R1
0467	AND	002	PL
0470	STO	107	R17
0474	LDA	014	LS7
0475	AND	015	LS8
0476	AND	016	LS9
0477	AND	017	LS10
0500	AND	776	R1
0501	AND	002	PL
0502	STO	762	R18
0506	LCAC	773	R20
0507	AND	107	R17
0510	AND	047	TR6
0511	AND	762	R18
0512	STO	110	R19
0516	LDA	020	LS5
0517	AND	776	R1
0520	AND	002	PL
0521	STO	773	R20
0525	AUX	771	

VIPTRAN COMPILER, VERSION 3.00  
DATE: 09/03/73 TIME: 21.25.38.430

<OPTIONS> SOURCE NOSORT NOPRINTSORT CODE NOSORTFAIL MAP NOTTY NOSCRATCHPAD SKIPO

CARD#	STMT#	SOURCE STATEMENT
1	0	C
2	0	C PROGRAMMING EXAMPLE 14
3	0	C A PROGRAM TO ILLUSTRATE ERROR RECOVERY
4	0	C
5	0	C 'VIPTRAN' RECOVERS RATHER NICELY FROM A NUMBER OF SYNTACTIC
6	0	C AND SEMANTIC PROGRAMMING ERRORS. THIS EXAMPLE WILL ILLUSTRATE
7	0	C THE RANGE OF ERRORS DETECTED AND REPAIRED BY THE COMPILER.
8	0	C
9	0	C PROGRAMMER: ALFRED C. WEAVER
10	0	C
11	0	C
12	0	C MISSING DECLARATION KEYWORD BEFORE VARIABLES
13	0	C
14	0	C VARIABLE1 VARIABLE2
*ON CARD #	14*	UNRECOGNIZABLE VIPTRAN SECTION NAME: VARIABLE1
*ON CARD #	14*	UNRECOGNIZABLE VIPTRAN SECTION NAME: VARIABLE2
15	0	C
16	0	C TIMER DEFINITION
17	0	C
18	0	C TIMER. T1 40 : T5 (T5INPUT)
*ON CARD #	18*	MISSING "(" IN TIMER DEFINITION OF: T1
*ON CARD #	18*	ILLEGAL VIPTRAN VARIABLE NAME: :
19	0	C
20	0	C TIMER. T8 42 (T8INPUT 61 T7 (T7INPUT)
*ON CARD #	20*	MISSING ")" IN TIMER DEFINITION
21	0	C
22	0	C ADDRESSING
23	0	C
24	0	C OUTPUTAC. R1 600 R2 600 R3 1006 R4 0 R5 250
*ON CARD #	24*	THIS ADDRESS USED MORE THAN ONCE: 600
*ON CARD #	24*	OCTAL ADDRESS > 777 FOR VARIABLE: R3
*ON CARD #	24*	OCTAL ADDRESS = 0 FOR VARIABLE: R4
25	0	C
26	0	C SCRATCH. S1 1020 S2 666 S3 2010 S4 1000 S5 1021
*ON CARD #	26*	OCTAL ADDRESS < 1001 FOR SCRATCHPAD VARIABLE: S2
*ON CARD #	26*	OCTAL ADDRESS > 1777 FOR SCRATCHPAD VARIABLE: S3
*ON CARD #	26*	OCTAL ADDRESS < 1001 FOR SCRATCHPAD VARIABLE: S4
27	0	C
28	0	C LATCH. L1 107 L2 108
*ON CARD #	28*	THIS ADDRESS IS NOT OCTAL: 108
29	0	C
30	0	C MISMATCHED VARIABLE TYPES ON A SINGLE VIP CARD
31	0	C ARE FLAGGED LATER, WHEN VIP CARDS ARE ALLOCATED
32	0	C
33	0	C INPUT12. INP12 100 INPUT24. INP24 101
34	0	C
35	0	C VIPTRAN ACCEPTS REPEATED, BUT UNAMBIGUOUS, ADDRESSES
36	0	C
37	0	C TIMER. T2 (R5 250)
38	0	C T3 41 (S5)
39	0	C
40	0	C

```

41      0      C
42      0      PROGRAM.
43      1      C
44      1      * C   VARIOUS SYNTAX ERRORS ARE POSSIBLE WITH EXPRESSIONS:
45      1      C
46      1      C   MISSING ASSIGNMENT OPERATOR
47      1      C
48      1      W   X * Y * Z
*IN STMT # 1* MISSING "=" INSERTED BEFORE: X
49      2      C
50      2      C   ILLEGAL CHARACTERS
51      2      C
52      2      A = F?
*IN STMT # 2* THIS ILLEGAL CHARACTER SKIPPED: ?
53      3      C
54      3      C   ILLEGAL NAMES
55      3      C
56      3      B = NAME-IS-LONGER-THAN-12-CHARACTERS
*IN STMT # 3* NAME TOO LONG - FIRST 12 CHARACTERS USED FOR: NAME-IS-LONGER-THAN-12-CHARACTERS
57      4      C
58      4      C   MISUSE OF COLUMN 1
59      4      C
60      4      X = Y*Z
*IN STMT # 4* ILLEGAL USE OF COLUMN 1 - CARD SKIPPED
61      4      C
62      4      C   UNBALANCED PARENTHESES
63      4      C
64      4      A = B + (C * ((D + E) * F)))
*IN STMT # 4* EXTRA ")" DELETED
65      5      C
66      5      B = ((P + C) * (D + E))
*IN STMT # 5* MISSING ")" SUPPLIED AT END OF STATEMENT
67      6      C
68      6      C   MISSING VARIABLES
69      6      C
70      6      C = D * + E * F
*IN STMT # 6* MISSING OPERAND - "DUMMY" INSERTED BEFORE: +
71      7      C
72      7      C   MISSING OPERATORS
73      7      C
74      7      D = E F G
*IN STMT # 7* MISSING OPERATOR - "*" SUPPLIED BEFORE: F
*IN STMT # 7* MISSING OPERATOR - "*" SUPPLIED BEFORE: G
75      8      C
76      8      E = (F+G) (H+K)
*IN STMT # 8* MISSING OPERATOR - "*" SUPPLIED BEFORE: (
77      9      C
78      9      C   ILLEGAL TIMER FUNCTIONS
79      9      C
80      9      F = T2-X00 * T2-XXX * T9-OX0
*IN STMT # 9* INVALID TIMER FUNCTION - THIS IS NOT A PROPER FUNCTION: T2-XXX
*IN STMT # 9* INVALID TIMER FUNCTION - THIS VARIABLE IS NOT A TIMER: T9
81      10     C
82      10     C   VIOLATION OF SYNTAX FOR SPECIAL OPERATORS
83      10     C
84      10     G = (B+C) & (D+E) : Q
*IN STMT # 10* ILLEGAL USE OF "&" OR ":" OPERATOR
*IN STMT # 10* ILLEGAL USE OF "&" OR ":" OPERATOR
85      11     C
86      11     C   MISSING '$TRANSLATE' CARD

```

```
      87      11      C
*IN STMT # 11* END OF SOURCE PROGRAM - "$TRANSLATE" INSERTED
      88      11      $TRANSLATE
*AFter STMT # 11* INCOMPATIBLE VARIABLE TYPE AND VIP CARD TYPE FOR: INP12
*AFter STMT # 11* INCOMPATIBLE VARIABLE TYPE AND VIP CARD TYPE FOR: INP24
```

## APPENDIX B

VIPTRAN Language Syntax, Modified  
Backus-Naur Form

To avoid ambiguity between parentheses as operators and as BNF metasymbols, parentheses used as operators will be underlined. Character strings are enclosed in single quotes.

```

<batch> ::= <program>*

<program> ::= <$VIP card> <declaration segment>
               <program segment>

<$VIP card> ::= ' $VIP ' (<default option> | <non-default
               option>)*

<default option> ::= 'SOURCE' | 'NOSORT' | 'NOPRINTSORT' | 'CODE' |
               'NOSORTFAIL' | 'MAP' | 'TTY' | 'NOSCRATCHPAD' |
               'SKIPO'

<non-default option> ::= 'NOSOURCE' | 'SORT' | 'PRINTSORT' | 'NOCODE' |
               'SORTFAIL' | 'NOMAP' | 'NOTTY' | 'SCRATCHPAD' |
               'SKIP' <decimal digit>

<declaration segment> ::= (<declaration keyword>{<variable name>
               [<octal address>]}) <timer declaration>*

<declaration keyword> ::= 'INPUT6.' | 'INPUT12.' | 'INPUT24.' | 'INPUT48.' |
               'INPUT120.' | 'OUTPUTAC.' | 'OUTPUTDC.' |
               'LATCH.' | 'SCRATCH.'

<timer declaration> ::= 'TIMER.'{<variable name>[<octal address>]
               ([<variable name>[<octal address>]])}

<program segment> ::= 'PROGRAM.' <assignment statement>*
               <$TRANSLATE card> <$END card>

<assignment statement> ::= <variable name> = <expression>

<expression> ::= <term>{+ <term>}

<term> ::= <factor>{* <factor>}

<factor> ::= <identifier> | (<expression>)

<identifier> ::= [/] (<variable name> | <timer function>)

<timer function> ::= <variable name>(' -OOX' | ' -OXO' | ' -OXX' | ' -XXO' |
               ' -XOX' | ' -XXO' )

```

<variable name>	::=	<alphabetic>{< alphanumeric>} <sub>0</sub> <sup>11</sup>
<alphabetic>	::=	'A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J'  'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T'  'U' 'V' 'W' 'X' 'Y' 'Z' ' ' '.' '-'
<alphanumeric>	::=	<alphabetic> <decimal digit>
<decimal digit>	::=	<octal digit> '8' '9'
<octal digit>	::=	'0' '1' '2' '3' '4' '5' '6' '7'
<octal address>	::=	<octal digit>{<octal digit>} <sub>0</sub> <sup>3</sup>
<\$TRANSLATE card>	::=	'\$TRANSLATE' in card columns 1-10
<\$END card>	::=	'\$END' in card columns 1-4

## APPENDIX C

## VIPTRAN Error Messages

This appendix lists all of the VIPTRAN error messages produced by the compiler. Each entry shows

- (1) the English text of the message;
- (2) the physical location of the error message on the source listing ("where");
- (3) an explanation of the circumstances surrounding, causing, or affecting the error ("why");
- (4) one or more examples of an instance of this particular error with an explanation of what is wrong ("example");
- (5) the possible steps the user should take to fix the error ("solution").

Errors not found in this appendix do not arise from the compiler; consult a systems programmer.

UNRECOGNIZABLE VIPTRAN SECTION NAME: <name>

where: Declaration segment, after \$VIP card and before a recognizable declaration keyword such as INPUT6., INPUT12., INPUT24., INPUT48., INPUT120., OUTPUTAC., OUTPUTDC., LATCH., TIMER., SCRATCH., or PROGRAM. <name> is the unrecognized name.

why: A character string which qualifies as a variable name appears before the first declaration keyword.

example: \$VIP

A B C

INPUT6. D E F

Variables A, B, and C are in error because they are not preceded by a type declaration keyword.

example: \$VIP

INPUT100. A

INPUT120. B

Variables INPUT100. and A are not preceded by a type declaration keyword. INPUT100. is a legal variable name but is not a type declaration keyword.

solution: Prefix all variables in a block with a legal type declaration keyword.

MISSING "(" IN TIMER DEFINITION OF: <name>

where: Declaration segment, in a TIMER. declaration block.

<name> is the name of the timer.

why: In the declaration, all timer names are followed by an optional address, a left parenthesis, the timer's input variable name, an optional address, and a right parenthesis.

example: TIMER. T1 T1-INPUT  
T1 is not followed by "(".

example: TIMER. T2 40 T2.IN  
40 is not followed by "(".

solution: For the above cases, use TIMER. T1 (T1-INPUT)  
T2 40 (T2.IN)

ILLEGAL VIPTRAN VARIABLE NAME: <name>

where: Declaration segment.

why: The compiler did not find a legal variable name where one was expected. <name> is the illegal quantity.

example: OUTPUTAC. A 20 21 C 23

A variable is missing between 20 and 21.

solution: Check sequence of variables and addresses.

## MISSING ")" IN TIMER DEFINITION

where: Declaration segment, in a TIMER. declaration block.

why: In the declaration, all timer names are followed by an optional address, a left parenthesis, the timer's input variable name, an optional address, and a right parenthesis.

example: TIMER. T1 (T1-INPUT T2 (T2.IN)  
missing ")" after T1-INPUT

example: TIMER. T2 40 (T2.IN 60 T3 (T3.IN)  
missing ")" after 60

solution: Add missing right parenthesis.

THIS ADDRESS USED MORE THAN ONCE: <address>

where: Declaration segment.

why: A single address has been assigned to two different variable names.

example: INPUT12. A 1 B 2 C 3

OUTPUTAC. X 2 Y 21 Z 22

Address 2 is used for both B and X.

solution: (1) If the associated variable is of type OUTPUTAC, OUTPUTDC, or SCRATCH, and does not control a physical device, omit the declaration and let the compiler assign an unused address.

(2) Reassign one of the variables to an unused address.

OCTAL ADDRESS < 1001 FOR SCRATCHPAD VARIABLE: <name>

where: Declaration segment, in a SCRATCH. declaration block.

why: Scratchpad variables must have addresses in the range  
 $1001_8 \leq \text{<address>} \leq 1777_8$ .

example: SCRATCH. X 600 Y 601  
 Addresses 600 and 601 are out of range for scratchpad  
 variables X and Y.

example: SCRATCH. Q1 1000 Q2 1001  
 Address  $1000_8$  is reserved and can not be assigned by the user.

example: SCRATCH. X 1010  
 OUTPUTCA. Y 3  
 Keyword "OUTPUTAC." is mis-spelled, making "OUTPUTCA."  
 a legal scratchpad variable with no address assigned; now Y,  
 an output variable, is within the SCRATCH declaration block  
 and 3 is an illegal address.

solution: (1) Keep scratchpad addresses in range.  
 (2) Check spelling of declaration keywords.

OCTAL ADDRESS > 1777 FOR SCRATCHPAD VARIABLE: <name>

where: Declaration segment, in a SCRATCH. declaration block.

why: <address> is out of range of machine.

example: SCRATCH. S 2000

$2000_8 > 1777_8$ , the largest machine address.

solution: (1) Omit address and allow compiler to assign an  
unused scratchpad address;

(2) Use addresses in the range  $1001_8 \leq \text{<address>} \leq 1777_8$ .

OCTAL ADDRESS = 0 FOR VARIABLE: <name>

where: Declaration segment.

why: <name> has been assigned an address of 0; 0 is reserved  
and can not be assigned by the user.

example: INPUT6. A 0 B 1 C 2

solution: Change 0 to an unused address.

OCTAL ADDRESS > 777 FOR VARIABLE: <name>

where: Declaration segment.

why: The address assigned to <name> is out of range of the machine for a non-scratchpad variable.

example: OUTPUTDC. X 776 Y 1777

Y is not type SCRATCH, so 1777 is out of range.

example: OUTPUTDC. Z 777

SCRATHC. A 1001 B 1002

Keyword "SCRATCH." is mis-spelled, making "SCRATHC." a legal OUTPUTDC variable with no address assigned. Variables A and B, which are in scratchpad, have joined the OUTPUTDC. declaration, making addresses 1001 and 1002 out of range.

solution: (1) Keep addresses in range.

(2) Check spelling of SCRATCH. keyword.

MISSING "=" INSERTED BEFORE: <name>

where:        Program segment.

why:        Assignment statements are of the form  
                                 <variable> = <expression>

The "=" was not found.

example:    A B\*C+D  
             Missing = between A and B.

solution:    insert = between <variable> and <expression>

STATEMENT DOES NOT BEING WITH VARIABLE NAME

where:        Program segment.

why:         Column 1 was blank, indicating the beginning of an  
assignment statement, but the first item on the card  
was not a variable name.

example:      $6A = B + C$   
"6A" is an illegal variable name.

example:      $A = B * C * D + E * F$   
\*G  
Second card is a continuation but column 1 is blank.

solution:    (1) Assure that variable names are legal.  
(2) Check column 1 for continuation character ">" if this  
is a continuation card.

THIS ADDRESS IS NOT OCTAL: <address>

where: Declaration segment.

why: Programmer-supplied address is not an octal number.

example: LATCH. X 76 Y 77 Z 78  
<address> = 78 is not octal.

solution: Use octal addresses only.

END OF SOURCE PROGRAM - "\$TRANSLATE" INSERTED

where:       Program segment.

why:         The card after the last assignment statement was not a  
\$TRANSLATE card, with those 10 characters in columns 1-10.

The program segment format is:

PROGRAM.

{all assignment statements}

\$TRANSLATE

\$END

example:     PROGRAM.

{all assignment statements}

A = B\*C+D     last card

solution:    Place \$TRANSLATE and \$END cards after the last  
assignment statement.

## ILLEGAL USE OF COLUMN 1 - CARD SKIPPED

where: Anywhere

why: A non-recognized code was used in column 1. The only legal codes are:

c comment

> continuation

\$ \$TRANSLATE, \$END cards in  
PROGRAM segment

blank begins new assignment statement

example: PROGRAM.

A = B\*

< (C+D)

Column 1 should have been the continuation character ">".

solution: Verify that column 1 contains one of the 4 legal codes. Note that the entire source card is skipped (just like a comment) when this error occurs.

THIS ILLEGAL CHARACTER SKIPPED: <character>

where: Anywhere

why: An individual character has been found which is not a VIPTRAN variable name, operator, or delimiter.

example: OUTPUTDC. A B C\$D

The "\$" is illegal as an alphabetic character. The compiler will give the error and use OUTPUTDC variables A, B, C, and D.

example: POWER = GATE\*POWER-SUPPLY + MANUAL?

The "?" is an illegal alphabetic character.

solution: (1) Verify correct spelling of variable names.  
(2) Remove offending character.

NAME TOO LONG - FIRST 12 CHARACTERS USED FOR: <name>

where: Anywhere

why: Variable names contain 12 or fewer characters. A character string longer than 12 characters has been used as a variable name or timer function.

example: OUTPUTDC. MASTERPOWEROUT 100  
The name "MASTERPOWEROUT" is too long.

example: TIMER. INTERVAL.TIM 40 (TIM 20)  
PROGRAM.  
MASTER = POWER \* INTERVAL.TIM-OXO  
Although "INTERVAL.TIM" is a legal timer name (12 characters), its use in a function makes a name 16 characters long. Timer functions have a 12 character limit just like variable names.

example: LATCH. INPUT21OUTPUT22LIGHT23  
No delimiters (spaces) are found between variables and addresses.  
The entire string becomes one variable "INPUT21OUTPU".

solution: (1) Rename offending variable.  
(2) Check length of timer function names.  
(3) Use spaces freely between variables and addresses.

## SERIOUS ERROR(S) PREVENT FURTHER PROCESSING

where:       After \$TRANSLATE card.

why:         An error or errors of such severity that continued  
              compilation is meaningless have occurred. The compiler  
              halts.

example:     Syntax errors in the PROGRAM segment make code generation  
              meaningless.

solution:    Correction of all other errors will clear this one also.

## SORT ROUTINE HAS FAILED

where:       After \$TRANSLATE card.

why:         The sort routine, as invoked by the SORT compiler option, has failed to reorder the equations in a way which makes them independent.

example:         PROGRAM.

1           A = B\*C

2           B = A\*C

3 \$TRANSLATE

\*\*\*AFTER STMT #3 \*\*\* SORT ROUTINE HAS REMOVED STMT #2

\*\*\*AFTER STMT #3 \*\*\* SORT ROUTINE HAS FAILED

Equations 1 and 2 are interlocked because the evaluation of A requires knowledge of B, yet evaluation of B requires knowledge of A. The programmer must decide if this situation is dangerous.

solution:   (1) Correct equations to remove interdependencies, if possible.

            (2) If situation is not dangerous, run again using NOSORT option.

## TERMINAL ERROR ENCOUNTERED DURING CODE GENERATION

where:       After \$TRANSLATE card.

why:         A serious error during code generation makes further processing impossible.

example:     A program has only one unused position on an OUTPUT card, and the NOSCRATCHPAD option is in effect. The compiler attempts to translate

$$A = B * C + D * E + F * G$$

which requires two temporary variables. When the second one is needed and no machine address is available, the error occurs.

solution:    This error occurs in conjunction with another more specific error preceding this one. Clearing the specific error(s) clears this one also.

## COMPILER ERROR - SYMBOL TABLE OVERFLOW

where:       Anywhere

why:         For any one program, the compiler can only recognize a fixed total number of programmer-defined and compiler-defined variables. The current limit is 600.

example:     The sum of unique variable names, timer functions, and reserved words (currently 23) in one program exceeds 600.

solution:    (1) Assure that the PROGRAM segments of two or more programs have not been intermixed.  
              (2) Call a systems programmer.

## COMPILER ERROR - POSTFIX STACK OVERFLOW

where:       Program segment.

why:         For any one program, the compiler can only store a fixed number of symbols which constitute an internal version of the source. The current limit is 3000.

example:     For programs with a large number of statements and/or with many lengthy assignment statements, the internal representation may exceed 3000 symbols.

solution:    (1) Assure that the PROGRAM segments of two or more programs have not been intermixed.  
              (2) Call a systems programmer.

EXTRA ")" DELETED

where:       Program segment.

why:        An assignment statement contains an unbalanced number of left and right parentheses (the number of each must be equal). Either too few left parentheses or too many right parentheses were included.

example:     $A = B + (C*((D+E)*F))$

There are 3 left parentheses and 4 right parentheses.

solution:   (1) Count number of left and right parentheses; assure proper balance.

(2) Be sure the source statement is entirely within columns 2-72 (columns 73-80 are not examined).

## COMPILER ERROR - PARSER STACK OVERFLOW

where:        Program segment.

why:         An assignment statement is so deeply nested that the parser stack overflows.

example:      $A = (B+(C+(D+(E+(F+(G+(H+(I))))))))$

Although the statement is syntactically correct, the equation is nested 8 levels deep and the parser stack overflows during evaluation.

solution:    (1) Determine whether the equation can be re-written to use fewer levels of nesting.

              (2) Call a systems programmer.

MISSING OPERAND - "DUMMY" INSERTED BEFORE: <operator>

where: Program segment.

why: In an expression, two operators were detected with no variable between them.

example:  $A = B*+D$

Missing operand between \* and +.

example:  $A = /B*/+C$

Missing operand between / and +.

solution: (1) Supply missing operand.

(2) Insure that only columns 2-72 were used for the source text. A variable in columns 73-80 would be lost.

(3) If the equation is continued, assure that the following card contains the continuation character ">" in column 1. Comments may not separate a source card and its continuation card.

MISSING ")" SUPPLIED AT END OF STATEMENT

where:        Program segment.

why:        An assignment statement contains an unbalanced number of left and right parentheses (the number of each must be equal). Either too many left parentheses or too few right parentheses were included.

example:     $A = ((B+C)*(D+E))$

There are 3 left parentheses and 2 right parentheses.

solution:   (1) Count number of left and right parentheses; assure proper balance.

             (2) Be sure the source statement is entirely within columns 2-72 (columns 73-80 are not examined).

## MISSING OPERATOR - "\*" SUPPLIED

where:       Program segment.

why:         In an expression, two operands were detected with no operator between them.

example:     A = B+C D  
Missing operator between C and D.

solution:    (1) Supply missing operator.  
              (2) Insure that only columns 2-72 were used for the source text. An operator in columns 73-80 would be lost.  
              (3) If the equation is continued, assure that the following card contains the continuation character ">" in column 1. Comments may not separate a source card and its continuation card.

INVALID TIMER FUNCTION - THIS VARIABLE IS NOT A TIMER: <variable>

where: Program segment.

why: The variable name <variable> was not declared to be  
of type TIMER.

example: INPUT12. IN1 IN2  
OUTPUTDC. OUT1 OUT2  
PROGRAM.

OUT1 = IN1 \* IN2 \* T1-0X0

OUT2 = IN1 \* IN2 + T1-X00

Variable T1 has not been declared to be a TIMER variable.

solution: Declare <variable> to be a timer using the format

TIMER. <timer variable name> <optional address>

(<timer output variable name> <optional address>)

INVALID TIMER FUNCTION - THIS IS NOT A PROPER FUNCTION: <name>

where:       Program segment.

why:         The suffix of a timer variable is not one of the six  
legal functions.

example:     TIMER. TIMER1 (T1-IN)  
  
PROGRAM.  
  
A = TIMER1-OOX \* TIMER1-XXX  
  
Suffix -XXX is not legal.

solution:    (1) Insure that the functional suffix is one of the six  
legal functions: -OOX, -OXO, -OXX, -XOO, -XOX, -XXO.  
  
              (2) Insure that the functional suffix uses the letter O  
and not the digit zero.  
  
              (3) Insure that the functional name does not extend  
beyond column 72.

INCOMPATIBLE VARIABLE TYPE AND VIP CARD TYPE FOR: <variable>

where:       After \$TRANSLATE card.

why:         Variables of different type have programmer-supplied  
addresses on the same VIP card.

example:     INPUT6. IN1 1 IN2 2

INPUT12. IN3 3

Variable IN3 at location 3 is a 12-volt input. Addresses  
0-17<sub>8</sub> are 6-volt inputs from the INPUT6. declaration.

solution:    (1) Omit addresses and allow compiler to assign VIP cards  
and variable addresses.

             (2) Change address of <variable> to an unused location  
on a card of proper type.

NO AVAILABLE ADDRESS FOR: <name>

where:       After \$TRANSLATE card.

why:         There are no unused locations of the proper type for  
variable <name>.

example:     A program uses many types of variables and allows the  
compiler to assign addresses, and VIP card assignment  
used every card location in the VIP. The address  
assignment of <name> attempts to allocate a new VIP  
card and generates the error.

solution:    Use manual address assignment and the VIP combination  
cards. This will require adjustment of variable types  
and requires a systems programmer.

## COMPILER ERROR - TEMPORARY STACK OVERFLOW

where: After \$TRANSLATE card.

why: An equation requires more than 20 temporary variables to evaluate it, thus causing an overflow of the temporary stack.

example: 
$$A = (B+C)*(D+E)*(F+G)*(H+I)*(J+K) \\ > \quad *(L+M)*(N+O)*(P+Q)*(R+S)*(T+U) \\ > \quad *(V+W)*(X+Y)*(Z+A1)*(B1+C1)*(D1+E1) \\ > \quad *(F1+G1)*(H1+I1)*(J1+K1)*(L1+M1)*(N1+O1) \\ > \quad *(P1+Q1)$$

Each OR pair is stored in a temporary variable so that a field engineer can examine its content. The compiler can use a maximum of 20 temporary variables to evaluate one expression.

solution: (1) Rewrite the equation to use fewer temporaries.

(2) Break one long equation into two shorter ones, introducing one virtual control relay as in:

original:  $Z = \langle \text{first half} \rangle * \langle \text{second half} \rangle$

new:  $VCR = \langle \text{first half} \rangle$

$Z = VCR * \langle \text{second half} \rangle$

## MEMORY OVERFLOW - TOO MANY TEMPORARIES

where: After \$TRANSLATE card.

why: The number of temporary locations needed by an equation exceeds the total number of unused locations on all OUTPUTAC. and OUTPUTDC. cards.

example: OUTPUTAC. A1 A2 A3 A4 A5 A6 A7 A8  
A9 A10 A11 A12 A13 A14 A15 A16

PROGRAM.

$$A = (B+C)*(D+E)$$

The generation of a temporary location to save (B+C) fails because there are no unused locations on the (one) card of type output.

solution: Rerun the program using the SCRATCHPAD option.

## ILLEGAL USE OF "&amp;" OR ":" OPERATOR

where:        Program segment.

why:        The "&" and ":" operators may have as their  
            left-hand side argument only a single variable name  
            (not an expression).

example:    A = B\*&D

solution:    Introduce a new variable to hold the controlling  
            expression and use the new variable as the argument  
            of "&" or ":". For the above example, use

            MASTER = B\*C

            A = MASTER & D

## COMPILER ERROR - TOO MANY STATEMENTS

where:       Program segment.

why:         The compiler will accept only a fixed number of equations  
              in any one program. That limit is currently 512.

example:     An exceptionally long program uses more than 512  
              assignment statements and generates the error.

solution:    (1) Assure that the program segments of two or more  
              programs have not been intermixed.  
              (2) Call a systems programmer.

## UNUSED OUTPUTS PROVIDE INSUFFICIENT TEMPORARIES

where:       After \$TRANSLATE card.

why:         There are not enough total unused locations on OUTPUTAC  
and OUTPUTDC cards to supply all the temporaries needed  
to evaluate an expression.

example:     OUTPUTAC.  A1 A2 A3 A4 A5 A6 A7 A8  
                  A9 A10 A11 A12 A13 A14 A15 A16

PROGRAM.

$A = (B+C)*(D+E)$

The temporary needed to hold (B+C) can not be assigned  
to the (only) OUTPUTAC card because the card is full.

solution:    Rerun the program using the SCRATCHPAD option.

## RERUN PROGRAM USING "SCRATCHPAD" OPTION

where:       After \$TRANSLATE card.

why:         The NOSCRATCHPAD option was in effect when the compiler could not find an unused location of the proper type for a necessary compiler-generated temporary location.

example:     At a time when all addresses on OUTPUTAC and OUTPUTDC cards are in use, the compiler needs another temporary. Module TEMPGEN fails and produces this message.

solution:    This message is not really an error, but a helpful hint. Utilizing the SCRATCHPAD option will allow 512<sub>10</sub> new variable locations and possibly free some OUTPUT locations - all temporary variables are allocated in scratchpad.

SORT ROUTINE HAS REMOVED STMT # <number>

where:       After \$TRANSLATE card.

why:       The SORT option was invoked. The sorter found one or more interlocked equations (they could not be reordered to make them independent). The sorter dropped statement number <number> (the largest statement number in the interlocked group) and attempts a resort.

example:   1       A = B\*C\*D  
             2       B = A\*B\*C  
             3       C = A+D+B  
             4       \$TRANSLATE

\*\*\*AFTER STMT #4 \*\*\* SORT ROUTINE HAS DROPPED STMT #3

\*\*\*AFTER STMT #4 \*\*\* SORT ROUTINE HAS DROPPED STMT #2

Equations 1, 2, and 3 are interlocked because A is a function of B and C, B is a function of A and C, and C is a function of A and B.

The first sorter pass drops equation 3. The remaining equations are still interlocked so statement 2 is dropped. The third pass succeeds trivially in sorting one equation.

solution:   Determine whether the interlocked equations represent a dangerous situation. If so, rewrite or reorder them. If not, rerun the program using the NOSORT option.



<b>BIBLIOGRAPHIC DATA HEET</b>	<b>1. Report No.</b> UIUCDCS-R-73-603	<b>2.</b>	<b>3. Recipient's Accession No.</b>
<b>Title and Subtitle</b>  VIPTRAN - A PROGRAMMING LANGUAGE AND ITS COMPILER FOR BOOLEAN SYSTEMS OF PROCESS CONTROL EQUATIONS			<b>5. Report Date</b> November 1973
<b>Author(s)</b> Alfred Charles Weaver			<b>6.</b>
<b>Performing Organization Name and Address</b> Department of Computer Science University of Illinois Urbana, Illinois 61801			<b>8. Performing Organization Rept. No.</b>
<b>Sponsoring Organization Name and Address</b> Department of Computer Science University of Illinois Urbana, Illinois 61801			<b>10. Project/Task/Work Unit No.</b>
			<b>11. Contract/Grant No.</b>
			<b>13. Type of Report &amp; Period Covered</b>
<b>Supplementary Notes</b>			<b>14.</b>
<b>Abstracts</b>  The solution of some industrial process control problems can be expressed as a system of Boolean algebra equations using variables representing inputs from and outputs to the real world. If this system is solved sequentially, cyclically, and in real time, the outputs may be used directly to control the state of real world devices. VIPTRAN is a high-level FORTRAN-like language which allows the programmer to express his Boolean system symbolically in logical (Boolean) equations. The VIPTRAN compiler reduces the input to machine code for the VIP process controller, and can plot the resulting system as a relay tree. The concept is extensible to other industrial controllers.			
<b>Key Words and Document Analysis. 17a. Descriptors</b>			
<b>Identifiers/Open-Ended Terms</b>			
<b>COSATI Field/Group</b>			
<b>Availability Statement</b>		<b>19. Security Class (This Report)</b> UNCLASSIFIED	<b>21. No. of Pages</b> 162
		<b>20. Security Class (This Page)</b> UNCLASSIFIED	<b>22. Price</b>

JAN 22 1974











JUN 6 1974



UNIVERSITY OF ILLINOIS-URBANA



3 0112 047417826